intel

APPLICATION NOTE

AP-153

June 1983

Designing with the 8256

Charles T. Yager
Applications Engineer

ORDER NUMBER: 210907-001

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

BXP, CREDIT, i, ICE, I^2ICE, ICS, iDBP, iDIS, iLBX, i_m, iMMX, Insite, INTEL, int_el, Intelevision, Intellec, $Int_eligent$ Programming, Intellink, iOSP, iPDS, iRMX, iSBC, iSBX, iSDM, iSXM, Library Manager, MCS, Megachassis, Micromainframe, MULTIBUS, Multichannel, Plug-A-Bubble, MULTIMODULE, PROMPT, Ripplemode, RMX/80, RUPI, System 2000, and UPI, and the combination of ICE, iCS, iRMX iSBC, MCS or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

* MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Department
3065 Bowers Avenue
Santa Clara, CA 95051

intel

Designing with the 8256

Contents

210907-001

Contents (cont.)

PAGE

INTRODUCTION

The INTEL 8256 MUART is a Multifunction Universal Asynchronous Receiver Transmitter designed to be used for serial asynchronous communication while also providing hardware support for parallel I/O, timing, counting and interrupt control. Its versatile design allows it to be directly connected to the MCS®-85, iAPX-86, iAPX-88, iAPX-186, and iAPX-188 microcomputer systems plus the MCS-48 and MCS-51 family of single-chip microcomputers.

The four commonly used peripheral functions contained in the MUART are:

1) Full-duplex, double-buffered serial asynchronous Receiver/Transmitter with an on-chip Baud Rate Generator
2) Two - 8-bit parallel I/O ports
3) Five - 8-bit counters/timers
4) 8-level priority interrupt controller

This manual can be divided into two parts. The first part describes the MUART in detail, including its functions, registers and pins. This section also describes the interface between the MUART and Intel CPUs plus a discussion on programming considerations. The second section provides an application example: a MUART-based line printer multiplexer. The Appendix contains software listings for the line printer multiplexer and some useful reference information.

DESCRIPTION OF THE MUART

The MUART can be logically partitioned into seven sections: the microprocessor bus interface, the command and status registers, clocking circuitry, asynchronous serial communication, parallel I/O, timer/event counters, and the interrupt controller. This can be seen from the block diagram of the 8256 MUART as shown in Figure 1. The MUART's pin configuration can be seen in Figure 2.

Microprocessor Bus Interface

The microprocessor bus interface is the hardware section of the MUART which allows a μP to communicate with the MUART. It consists of tristate bi-directional data-bus buffers, an address latch, a chip select ($\overline{CS}$) latch and bus control logic. In order to provide all of the MUART's functions in a 40-pin DIP while retaining direct register addressing, a multiplexed address/data bus is used.

Address/Data Bus

The MUART contains 16 internal directly addressable read/write registers. Four of the eight address/data lines are used to generate the address. When using 8-bit microprocessors such as MCS-85, MCS-48 and MCS-51, AD0 - AD3 are used to address the 16 internal registers while Address/Data line 4 (AD4) is not used for addressing. For 16-bit systems, AD1 - AD4 are used to generate the address for the internal data registers and AD0 is used as a second active low chip select.

$\overline{RD}$, $\overline{WR}$, $\overline{CS}$

The 8256 bus interface uses the standard bus control signals which are compatible with all Intel peripherals and microprocessors. The chip select signal ($\overline{CS}$), typically derived from an address decoder, is latched along with the address on the falling edge of ALE. As a result, chip select does not have to remain low for the entire bus cycle. However, the data bus buffers will remain tristated unless an $\overline{RD}$ or a $\overline{WR}$ signal becomes active while chip select has been latched in low.

INT, $\overline{INTA}$

The INT and $\overline{INTA}$ signals are used to interrupt the CPU and receive the CPU's acknowledgment to the interrupt request. The MUART can vector the CPU to the appropriate service routine depending on the source of the interrupt.

RESET

When a high level occurs on the RESET pin, the MUART is placed in a known initial state. This initial state is described under "Hardware Reset."

Command and Status Register

There are three command registers and one status register as shown in Figure 1. The three command registers are read/write registers while the status register is a read only. The command registers configure the MUART for its operating environment (i.e., 8 or 16 bits CPU, system clock frequency). In addition, they direct its higher level functions such as controlling the UART, selecting modes of operation for the interrupt controller, and choosing the fundamental frequency for the timers. Command Register 3 is the only register in the MUART which is a bit set/reset register, allowing the programmer to simply perform one write to set or reset any of the bits.

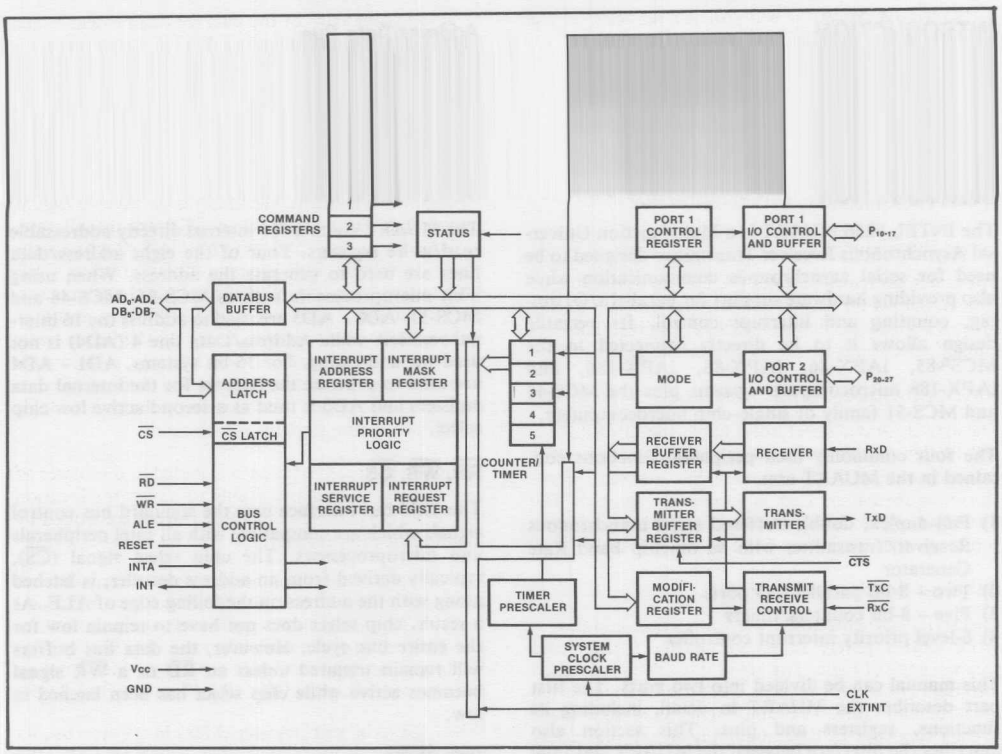


Figure 1. Block Diagram of the 8256 MUART

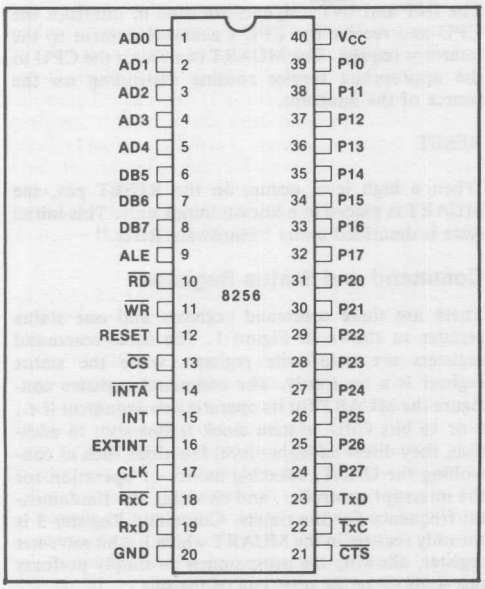


Figure 2. MUART Pin Configuration

The status register provides all of the information about the status of the UART's transmitter and receiver as well as the status of the interrupt pin. The status register is the only read only register in the MUART.

CLOCK CIRCUITRY

The clock for the five timers and baud rate generator is derived from the system clock. The system clock, pin 17 (CLK), is fed into a system clock prescaler which in turn feeds the five timers and the baud rate generator. The MUART's system clock can be asynchronous to the microprocessor's clock.

System Clock Prescaler

The system clock prescaler is a programmable divider which normalizes the internal clocking frequency for the timers and baud rate generator to 1.024MHz. It divides the system clock (CLK) by 1, 2, 3, or 5, allowing clock frequencies of 1.024MHz, 2.048MHz, 3.072MHz or 5.12MHz. (The commonly used 6.144MHz crystal frequency for the 8085 results in a 3.072MHz frequency from the 8085's CLK pin.) If the system clock is not one of the four frequencies mentioned above, then the frequency of the baud rate generator and the timers will be nonstandard;

however, the MUART will still run as long as the system clock meets the data sheet tcy spec.

Timer Prescaler

The timer prescaler permits the user to select one of two fundamental timing frequencies for all of the MUART's timers, either 1KHz or 16KHz. The frequency selection is made via Command Register 0.

Asynchronous Serial Interface

The asynchronous serial interface of the MUART is a full-duplex double-buffered transmitter and receiver with separate control registers. The standard asynchronous format is used as shown in Figure 3. The operation of the UART section of the MUART is very similar to the operation of the 8251A USART.

Receiver Section of the UART

The serial asynchronous receiver section contains a serial shift register, a receiver buffer register and receiver control logic. The serial input data is clocked into the receive shift register from the RxD pin at the specified baud rate. The sampling actually takes place at the rising edge of RxC, assuming an external clock,

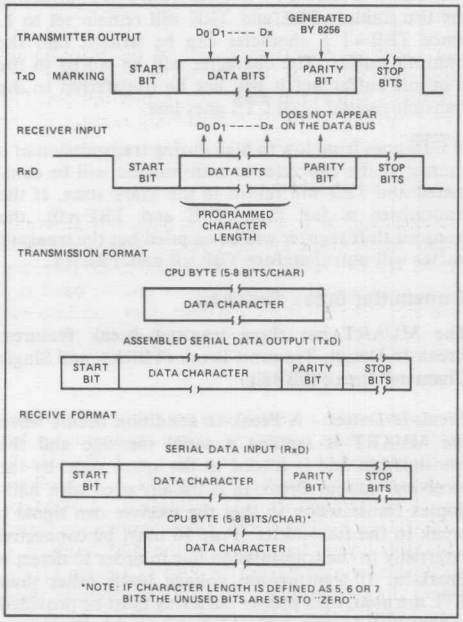


Figure 3. Asynchronous Format

or at the rising edge of the internal baud clock. When the receiver is enabled but inactive, the receive logic is sampling RxD at either 32 or 64 times the bit rate, looking for a change from the Mark (high) to the Space (low) state. This is commonly referred to as the start bit search mode. When this state change occurs, the receive logic waits one half of a bit time and then samples RxD again. If RxD is still in the Space state, the receive logic begins to clock in the receive data beginning one bit period later. If RxD has returned to the Mark state (i.e., false start bit), the receive logic will return to the start bit search mode.

Normally the received data is sampled in the center of each bit, however it is possible to adjust the location where the bit is sampled. This feature is controlled by the modification register.

The bit rate of the serial receive data is derived from either the internal baud rate generator or an external clock. When using an external clock, the programmer has a choice of three sampling rates: 1x, 32x, or 64x, Using the internal baud rate generator, the sampling rates are all 64x except for 19.2 Kbps which is 32x.

When the serial shift register clocks in the stop bit, an internal load pulse is generated which transfers the contents of the shift register into the receive buffer. This transfer takes place during the first half of the first stop bit. The load pulse also triggers several other signals relevant to the receive section including Receive Buffer Full (RBF), Parity Error (PE), Overrun Error (OE), and Framing Error (FE). These four status bits are updated after the middle of the first stop bit when the receive buffer has already been latched. Each one of these four status bits are latched. They are reset on the rising edge of the first read pulse (RD) addressed to the status register. A complete description of the status register is given in the section "Description of the Registers."

When the serial receiver is disabled (via bit 6 of Command Register 3) the load pulse is suppressed. The result is that the receive buffer is not loaded with the contents of the shift register, and the RBF, PE, OE, and FE bits in the status register are not updated. Even though the receiver is disabled, the serial shift register will still be clocking in the data from RxD, if any. This means that the receiver will still be synchronized with the start and stop bits. For example, if the receiver is enabled via Command Register 3 in the middle of receiving a serial character, the character will still be assembled correctly. When the receiver is disabled the last character received will remain in the receive buffer. On power-up the value in the receive buffer is undefined.

Whenever a character length of fewer than 8 bits is programmed, the most significant bits of a received character will read as zero. Also, the receiver will only check the first stop bit of any character, regardless of how many stop bits are programmed into the device.

Receive Break Detect

A Receive Break occurs when RxD remains in the space state for one character time, including the parity bit (if any) and the first stop bit. The MUART will set the Break Detect status bit (BD) when it receives a break. The Break Detect status bit is set after the middle of the first stop bit. If the MUART detects a break it will inhibit the receive buffer load pulse, thus the receive buffer will not be loaded with the null character, and none of the four status bits (PE, OE, FE, and RBF) will be updated. The last character received will remain in the receive buffer. A break detect state has the same effect as disabling the receiver—they both inhibit the load pulse—therefore one can think of the break status as disabling the receiver.

The Break Detect status bit is latched. It is cleared by the rising edge of the read pulse addressed to the status register. If a break occurs, and then the RxD data line returns to the Mark state before the status register is read, the BD status bit will remain set until it is read. If RxD returns to the Mark state after the BD status bit has been read true, the BD status bit will be reset automatically without reading the status register.

The receive break detect logic of the MUART is independent of whether the receiver is enabled or disabled; therefore even if the receiver is disabled the MUART will recognize a break. When the RxD line returns to the Mark state after a break, the 8256 will be in the start bit search mode.

If the receiver interrupt level is enabled, break will generate an interrupt request regardless of whether the receiver is enabled. Another receive interrupt will not be generated until the RxD pin returns to the Mark state.

Transmitter Section of the UART

The serial asynchronous transmitter section of the MUART consists of a transmit buffer, a transmit (shift) register, and the associated control logic. There are two bits in the status register which indicate the status of the transmit buffer and transmit register: TBE (transmit buffer empty) and TRE (transmit register empty).

To transmit a character, a byte is written to the transmit buffer. The transmit buffer should only be written to when TBE = 1. When the transmit register is empty and $\overline{CTS} = 0$, the character will be automatically transferred from the transmit buffer into the transmit register. The data transfer from the transmit buffer to the transmit register takes place during the transmission of the start bit. After this transfer takes place, sometime at the beginning of the transmission of the first data bit, TBE is set to 1.

When the transmitter is idle, both TBE and TRE will be set to 1. After a character is written to the transmit buffer, TBE = 0 and TRE = 1. This state will remain for a short period of time, then the character will be transferred into the transmit register and the status bits will read TBE = 1 and TRE = 0. At this point a second character may be written to the transmit buffer after which TBE = 0 and TRE = 0. TBE will not be set to 1 again until the transmit register becomes empty and is reloaded with the byte in the transmit buffer.

The transmitter can be disabled only one way—using the $\overline{CTS}$ pin. When $\overline{CTS} = 0$ the transmitter is enabled, and when $\overline{CTS} = 1$ the transmitter is disabled. If the transmitter is idle and $\overline{CTS}$ goes from 0 to 1, disabling the transmitter, TBE and TRE will remain set to 1. Since TBE = 1 a character can be written into the transmit buffer. The character will be stored in the transmit buffer but it will not be transferred to the transmit register until $\overline{CTS}$ goes low.

If $\overline{CTS}$ goes from low to high during transmission of a character, the character in transmission will be completed and TxD will return to the Mark state. If the transmitter is full (i.e., TBE and TRE = 0), the transmit shift register will be emptied but the transmit buffer will not; therefore TBE = 0 and TRE = 1.

Transmitter Break Features

The MUART has three transmit break features: Break-In Detect, Transmit Break (TBRK), and Single Character Break (SBRK).

Break-In Detect - A Break-In condition occurs when the MUART is sending a serial message and the transmission line is forced to the space state by the receiving station. Break-In is usually used with half-duplex transmission so that the receiver can signal a break to the transmitter. Port 16 must be connected externally to the transmission line in order to detect a Break-In. If transmission voltage levels other than TTL are used, then proper buffering must be provided so that Port 16 on the MUART will receive the correct polarity and voltage levels.

When Break-In Detect is enabled, Port 16 is polled internally during the transmission of the last or only stop bit of a character. If this pin is low during transmission of the stop bit, the Break Detect status bit (BD) will be set. Break-In Detect and receive Break Detect are OR-ed to set the BD status bit. (Either one can set this bit.) The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on level 5, the transmit interrupt, while Break will generate an interrupt on level 4, the receive interrupt. If RxC and TxC are used for the serial bit rates, Break-In cannot be detected.

Transmit Break – This causes the TxD pin to be forced low for as long as the TBRK bit in Command Register 3 is set. While Transmit Break is active, data transfers from the Transmit Buffer to the Transmit register will be inhibited.

If both the Transmit Buffer and the Transmit Register are full, and a Transmit Break command is issued (command register 3, TBRK = 1), the entire character in the Transmit register is sent including the stop bits. TxD is then driven low and the character in the Transmit Buffer remains there until Transmit Break is disabled (command register 3, TBRK = 0). At this time TxD will go high for one bit time and then send the character in the Transmit Buffer.

Single Character Break – This causes TxD to be set low for one character including start bit, data bits, parity bit, and stop bits. The user can send a specific number of Break characters using this feature.

If both the Transmit Buffer and the Transmit Register are full and a Send Break command is issued (command register 3, SBRK = 1) the entire character in the Transmit Register is sent including the stop bits. TxD is driven low for one complete character time followed by a high for two bit times after which the character in the Transmit Buffer is sent.

Modification Register

The modification register is used to alter two standard functions of the receiver (start bit check, and sampling time) and to enable a special indicator flag for half-duplex operation (transmitter status). Disabling start bit check means that the receiver will not return to the start bit search mode if RxD has returned to the Mark state in the center of the start bit. It will simply proceed to assemble a character from the RxD pin regardless of whether it received a false start bit or not. The modification register also allows the user to define where within the receive data bits the MUART will sample.

Parallel I/O

The MUART contains 16 parallel I/O pins which are divided into two 8-bit ports. These two parallel I/O ports (Port 1 and Port 2) can be used for basic digital I/O such as setting a bit high or low, or for byte transfers using a two-wire handshake. Port 1 is bit programmable for input or output, so any combination of the eight bits in Port 1 can be selected as either an input or an output. Port 2 is nibble programmable, which means that all four bits in the upper or lower nibble have to be selected as either inputs or outputs. For byte transfers using the two- wire handshake, Port 2 can either input or output the byte while two bits in Port 1 are used for the handshaking signals.

All of the bits in Port 1 have alternate functions other than I/O ports. As mentioned above, when using the byte handshake mode, two bits on Port 1 are used for the handshaking signals. As a result, these two bits cannot be used for general purpose I/O. The other six bits in Port 1 also have alternate functions if they are not used as I/O ports. Table 1 lists each bit from Port 1 and its corresponding alternate function.

The bits in the Port 1 Control Register select whether the pins on Port 1 are inputs or outputs. The pins on Port 1 are selected as control pins through the other programming registers which are relevant to the control signal. Configuring a bit in Port 1 as a control function overrides its definition in the Port 1 Control Register. If the pins on Port 1 are redefined as control signals, the definition of whether the pin is an input or an output in the Port 1 Control Register remains unchanged. If the pins on Port 1 are converted back to I/O pins, they assume the state which was defined in the Port 1 Control Register.

Each parallel I/O port has a latch and drivers. When the port is in the output mode, the data written to the port is latched and driven on the pins. The data which is latched in the I/O ports remains unchanged unless the port is written to again. Reading the ports, whether the port is an input or output, gates the state at the pins onto the data bus. Writing to an input port has no effect on the pin, but the data is stored in the latch and will be output if the direction on the pin is changed later. Writing to a control pin on Port 1 has the same effect as writing to an input pin. If pins 2, 3, 5, and 6 in Port 1 are used for control signals, the contents of the respective output latches will be read, not the state of the control signals. If pins 0, 1, and 7 on

Table 1. Port 1 Control Signals

Pin Symbol	Pin Number	Control Function	Condition
P10 P11	39 38	$\overline{ACK}$ Control signals for Port 2 $\overline{OBF}$ 8-bit handshake output	Mode register P2C2 - P2C0 = 101
P10 P11	39 38	$\overline{STB}$ Control signals for Port 2 $\overline{IBF}$ 8-bit handshake input	Mode register P2C2 - P2C0 = 100
P12	37	Event counter 2 clock input	Mode register CT2 = 1
P13	36	Event counter 3 clock input	Mode register CT3 = 1
P14	35	Internal baud rate generator clock output	Mode word P2C0 - P2C2 = 111 Port 1 control word P14 = 1 Command Register 2 B3 - B0 $\geq$ 3H
P15	34	Timer 5 trigger input	Mode register T5C = 1
P16	33	Break-In detection input	Command Register 1 BRKI = 1
P17	32	External edge sensitive interrupt input	Command Register 1 BITI = 1

Port 1 are used for control signals, the state of the control signals will be read. If pin 4 on Port 1 is used as a test output for the internal baud rate, this clock signal will be output through the output latch, thus the information in the output latch will be lost.

The Two-Wire Byte Handshake

The 8256 can be programmed, via the Mode Register, to implement an input or output two-wire byte handshake. When the Mode Register is programmed for the byte handshake, Port 2 is used to transmit or receive the byte, and pins P10 and P11 are used for the two handshake control signals. Figures 4 and 5 on pages 7 through 10 show a block diagram and timing signals for the two-wire handshake input and output.

To set up the two-wire handshake output using interrupts one must first program the Mode Register, and then enable the interrupt via the interrupt mask register. An interrupt will not occur immediately after the two-wire handshake interrupt is enabled. The interrupt is triggered by the rising edge of $\overline{ACK}$. There are two ways to generate the first interrupt. Either the first data byte must be written to Port 2 and completely transferred before an interrupt will occur, or the two-wire handshake interrupt is enabled while $\overline{ACK}$ is low, and then $\overline{ACK}$ goes high.

Event Counters/Timers

The MUART's five 8-bit programmable counters/timers are binary presettable down counters. The distinction between timer and counter is determined by the clock source. A timer measures an absolute time interval, and its input clock frequency is derived from the MUART's system clock. A counter's input clock frequency is derived from a pulse applied to an external pin. The counter is decremented on the rising edge of this pulse.

When the counters/timers are configured as timers their clock source passes through two dividers: the system clock prescaler, and the timer prescaler. As mentioned before, the system clock prescaler normalizes the internal system clock to 1.024 MHz. The timer prescaler receives this normalized system clock and devides it down to either 1 kHz or 16 kHz, depending

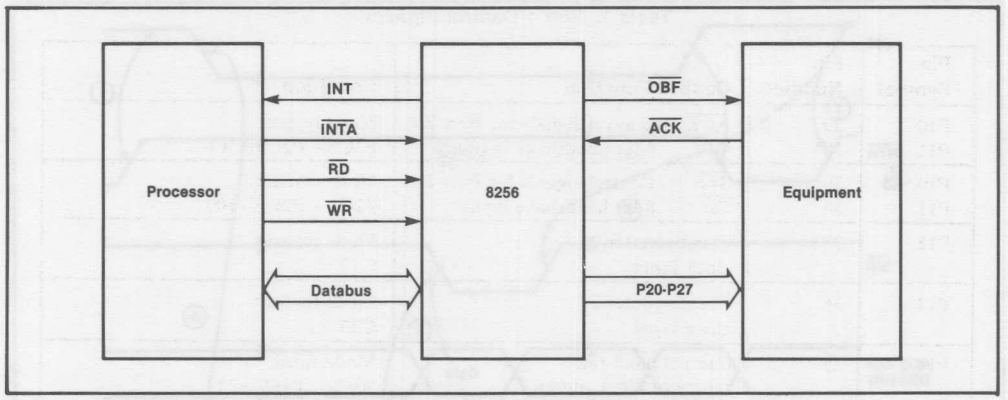


Figure 4. Block Diagram of Handshake Output

on how Command Register 1 is programmed. If more timing resolution is needed the clock frequency can be input externally through the I/O ports.

By programming the Mode Register, four of the 8-bit counters/timers can be cascaded to form two 16-bit counters. Counters/timers 3 and 5 can be cascaded together, and counters/timers 2 and 4 can be cascaded together. Counters/timers 2 and 3 are the lower bytes, while counters/timers 4 and 5 are the upper bytes in the cascaded mode.

Each counter can be loaded with an arbitrary initial value. Timer 5 is the only timer which has a special save register which holds its initial value. Whenever Timer 5 is loaded with an initial value the special save register is also loaded with this value. Timer 5 can be reloaded to its initial value from the detection of a high-to-low transition on Port P15.

The counters are decremented on the first rising edge of the clock after the initial value has been loaded. The setup time for loading the counter when using an external clock is specified in the data sheet. When using internal clocks, the user has no way of knowing the phase relationship of the clock to the write pulse; therefore the timing accuracy is one clock period.

The timers are counting continuously, and an interrupt request is issued any time a single counter or pair of cascaded counters reaches zero. If the timers are going to be used with interrupts, then the programmer should first load the timer with the initial value, then enable the interrupt. If the programmer enables the interrupt first, it is possible that the interrupt will occur before the initial value is loaded. When an interrupt from any one of the timers occurs, the corresponding bit in the interrupt mask register is automatically reset, preventing further interrupt requests from occuring.

The event counters/timers can be used in the following modes of operation:

Timer 1
— Serves as an 8-bit timer.

Event Counter/Timer 2
— Serves as an 8-bit timer or event counter, or cascaded with Timer 4 as a 16-bit timer or event counter.

Event Counter/Timer 3
— Serves as an 8-bit timer or event counter, or cascaded with Timer 5 as a 16-bit timer or event counter, with the additional modes of operation selectable for Timer 5.

Timer 4
— Serves as an 8-bit timer, or cascaded with Event Counter/Timer 2 as a 16-bit timer or event counter.

Timer 5
1) Non-retriggerable 8-bit timer
2) Retriggerable 8-bit timer whose initial value is loaded from a save register which starts following the negative transition of an external signal. Subsequent transitions of this signal after the counting has started, reloads the initial value and restarts the counting.
3) Cascaded with Event Counter/Timer 3, non-retriggerable 16-bit timer, which can be loaded with an initial value by two write operations.

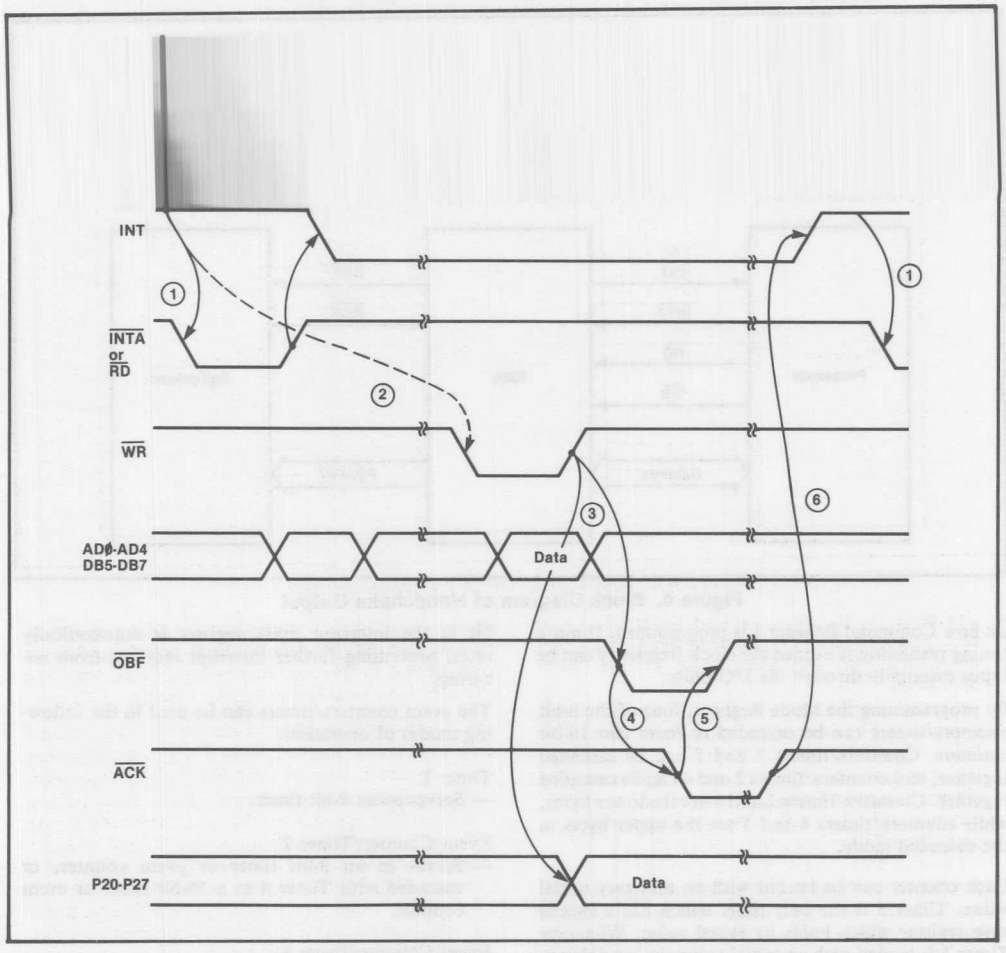


Figure 4a. Timing of Handshake Output

(1.) The 8256 signals with INT that the equipment has accepted the last character and that the output latches are empty again.

(2.) Thereupon, the microprocessor transfers the next data to the 8256.

(3.) The rising edge of WR latches the data into port 2 (P20...P27) and "Output Buffer Full" (OBF) is set which indicates that a new byte is available.

(4.) The equipment acknowledges with the falling edge of $\overline{ACK}$ that it recognized $\overline{OBF}$.

(5.) Thereupon, the 8256 releases $\overline{OBF}$.

(6.) The equipment acknowledges the data transfer with a rising edge of $\overline{ACK}$ which causes the 8256 to set INT.

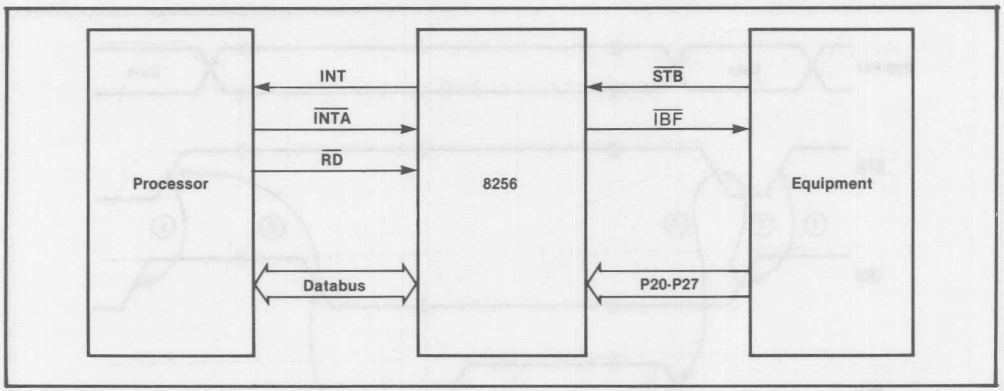


Figure 5. Block Diagram of Handshake Input

4) Cascaded with event counter/timer 3, non-retriggerable 16-bit event counter, which can be loaded with an initial value by two write operations.
5) Cascaded with Event Counter/Timer 3, retriggerable 16-bit timer. The most significant byte (Timer 5) will be loaded with its initial value from the save register, while the least significant byte (Event Counter/Timer 3) will be set to 0FFH automatically, Loading, starting, and retriggering operations follow the same pattern as in 2).
6) Cascaded with Event Counter/Timer 3, retriggerable 16-bit event counter. The most significant byte (Timer 5) will be loaded with its initial value from the save register, while the least significant byte (Event Counter/Timer 3) will be set to 0FFH automatically. Loading, starting, and retriggering operations follow the same pattern as in 2).

Interrupt Controller

In a microcomputer system there are several ways for the CPU to recognize that a peripheral device needs service. Two of the most common ways are the polling method and the interrupt service method.

In the polling method the CPU reads the status of each peripheral to determine whether it needs service. If the peripheral does not need service, the time the CPU spends polling is wasted; therefore this overhead results in increasing the execution time. Some systems must meet a specific request to response time such as a real time signal. In this case the programmer must guarantee that the peripheral is polled at a certain frequency. This polling frequency cannot always easily be met when the CPU must execute a main program as well as subroutines. Usually each peripheral has its own request to response time requirements; therefore the user must establish a priority scheme.

The interrupt method provides certain advantages over the polling method. When a peripheral device needs service it signals the CPU through hardware asynchronously, thus reducing the overhead of polling a device which does not need service. The CPU would typically finish the instruction it is executing, save the important registers, and acknowledge the peripheral's interrupt request. During the acknowledgment, the CPU reads a vector which directs the CPU to the starting location of the appropriate interrupt service routine. If several interrupt requests occur at the same time, special logic can prioritize the requests so that when the CPU acknowledges the interrupt, the highest priority request is vectored to the CPU.

An interrupt driven system requires additional hardware to control the interrupt request signal, priority, and vectoring. The 8256 integrates this additional hardware onto the chip. The interrupt controller on the MUART is directly compatible with the MCS-85, iAPX-86, iAPX-88, iAPX-186, iAPX-188 family of microcomputer systems, and it can also be used with other microprocessors as well. It contains eight priority levels, however, there are a total of 12 interruptable sources: 10 internal and 2 external. Since there are eight priority levels, only eight interrupts can be used at one time. The assignment of the interrupts used is selected by Command Register 1 and by the mode register. The MUART's interrupt sources have a fixed priority. Table 2 displays how the 12 interrupt sources are mapped into the 8 priority levels.

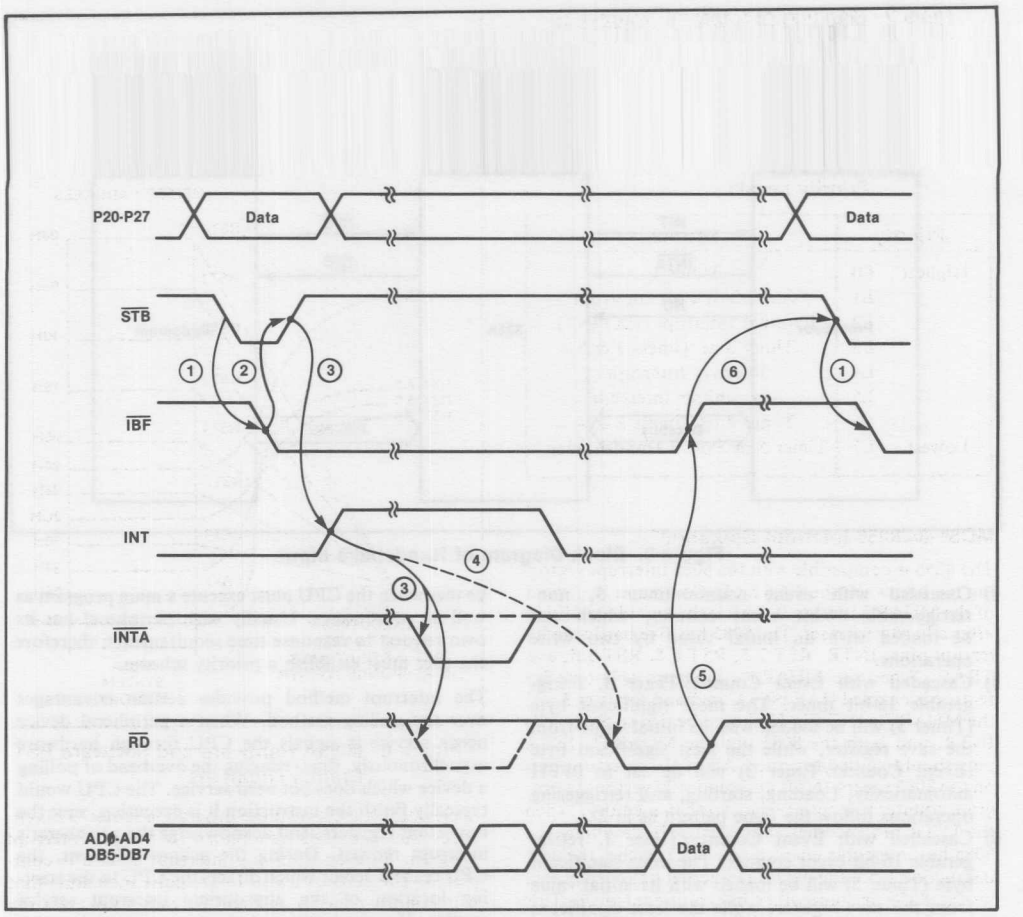


Figure 5a. Timing for Handshake Input

(1.) The equipment indicates with the falling edge of $\overline{STB}$ (Strobe) that a new character is available at port 2. The 8256 acknowledges the indication by activating $\overline{IBF}$ (Input Buffer Full).

(2.) Thereupon, the equipment releases $\overline{STB}$ and the 8256 latches the character.

(3.) The 8256 informs the microprocessor through INT that a new character is ready for transfer.

(4.) The microprocessor reads the character.

(5.) The rising edge of signal $\overline{RD}$ resets signal $\overline{IBF}$.

(6.) This action signals to the equipment that the input latches of the 8256 are empty and the next character can be transferred.

Table 2. Mapping of Interrupt Sources to Priority Levels

Priority		Source
Highest	L0	Timer 1
	L1	Timer 2 or Port Interrupt
	L2	External Interrupt (EXTINT)
	L3	Timer 3 or Timers 3 & 5
	L4	Receiver Interrupt
	L5	Transmitter Interrupt
	L6	Timer 4 or Timers 2 & 4
Lowest	L7	Timer 5 or Port 2 Handshaking

MCS®-85/8256 Interrupt Operation

The 8256 is compatible with the 8085 interrupt vectoring method when the 8086 bit in Command Register 1 of the MUART is set to 0. This is the default condition after a hardware reset. The 8085 has five hardware interrupt pins: INTR, RST 7.5, RST 6.5, RST 5.5, and TRAP. When the MUART's interrupt acknowledge feature is enabled (IAE bit 5 Command Register 3 = 1) the MUART's INT Pin 15 should be tied to the 8085's INTR, and both the 8085 and the MUART's INTA pins should be tied together. All of the interrupt pins on the 8085 except INTR automatically vector the program counter to a specified location in memory. When the INTR pin becomes active (HIGH), assuming the 8085 has interrupts enabled, the 8085 fetches the next instruction from the data bus where it has been placed by the 8256 or some other interrupt controller. This instruction is usually a Call or an RST0 through RST7. Figure 6 shows the memory locations where the 8085 will vector to based on which type of interrupt occurred.

The 8085 can receive an interrupt request any time, since its INTR input is asynchronous. The 8085, however, doesn't always acknowledge an interrupt request immediately. It can accept or disregard requests under software control using the EI (Enable Interrupt) or DI (Disable Interrupt) instructions.

At the end of each instruction cycle, the 8085 examines the state of its INTR pin. If an interrupt request is present and interrupts are enabled, the 8085 enters an interrupt machine cycle. During the interrupt machine cycle the 8085 automatically disables further interrupts until the EI instruction is executed. Unlike normal machine cycles, the interrupt machine

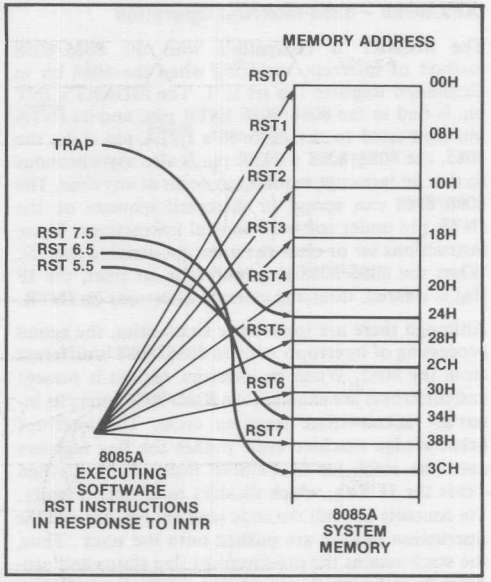


Figure 6. 8085A Hardware and Software RST Branch Locations

cycle doesn't increment the program counter. This ensures that the 8085 can return to the pre-interrupt program location after the interrupt service is completed. The 8085 issues an $\overline{\text{INTA}}$ pulse indicating that it is honoring the request and is ready to process the interrupt.

The 8256 can now vector program execution to the corresponding service routine. This is done during the first and only $\overline{\text{INTA}}$ pulse. Upon receiving the $\overline{\text{INTA}}$ pulse, the 8256 places the opcode RSTn on the data bus; where n equals 0 through 7 based on the level of the interrupt requested. The RSTn instruction causes the contents of the program counter to be pushed onto the stack, then transfers control to the instruction whose address is eight times n, as shown in Figure 6.

Note that because interrupts are disabled during the interrupt acknowledge sequence, the EI instruction must be executed in either the service routine or the main program before further interrupts can be processed.

For additional information on the 8085 interrupt operation and the RSTn instruction, refer to the *MCS-85 User's Manual.*

iAPX-86/88 - 8256 Interrupt Operation

The MUART is compatible with the 8086/8088 method of interrupt vectoring when the 8086 bit in Command Register 1 is set to 1. The MUART's INT pin is tied to the 8086/8088 INTR pin, and its $\overline{INTA}$ pin connected to the 8086/88's $\overline{INTA}$ pin. Like the 8085, the 8086/8088's INTR pin is also asynchronous so that an interrupt request can occur at any time. The 8086/8088 can accept or disregard requests on the INTR pin under software control instructions. These instructions set or clear the interrupt-enabled flag IF. When the 8086/8088 is powered-on or reset, the IF flag is cleared, disabling external interrupts on INTR.

Although there are some basic similarities, the actual processing of interrupts with an 8086/8088 is different from the 8085. When an interrupt request is present and interrupts are enabled, the 8086/8088 enters its interrupt acknowledge machine cycle. The interrupt acknowledge machine cycle pushes the flag registers onto the stack (as in PUSHF instruction). It then clears the IF flag, which disables interrupts. Finally, the contents of both the code segment register and the instruction pointer are pushed onto the stack. Thus, the stack retains the pre-interrupt flag status and program location which are used to return from the service routine. The 8086/8088 then issues the first of two INTA pulses which signals the 8256 that the 8086/8088 has honored its interrupt request.

The 8256 is now ready to vector program execution to the appropriate service routine. Unlike the 8085 where the first INTA pulse is used to place an instruction on the data bus, the first INTA pulse from the 8086/8088 is used only to signal the 8256 of the honored request. The second INTA pulse causes the 8256 to place a single interrupt vector byte onto the data bus. The 8256 places the interrupt vector bytes 40H through 47H corresponding to the level of the interrupt to be serviced. Not used as a direct address, this interrupt vector byte pertains to one of 256 interrupt "types" supported by the 8086/8088 memory. Program execution is vectored to the corresponding service routine by the contents of a specified interrupt type.

All 256 interrupt types are located in absolute memory locations 0 through 3FFH which make up the 8086/8088's interrupt vector table. Each type in the interrupt vector table requires 4 bytes of memory and stores a code segment address and an instruction pointer address. Figure 7 shows a block diagram of the interrupt vector table. When the 8086/8088 receives an interrupt vector byte, it multiplies its value by four to acquire the address of the interrupt type.

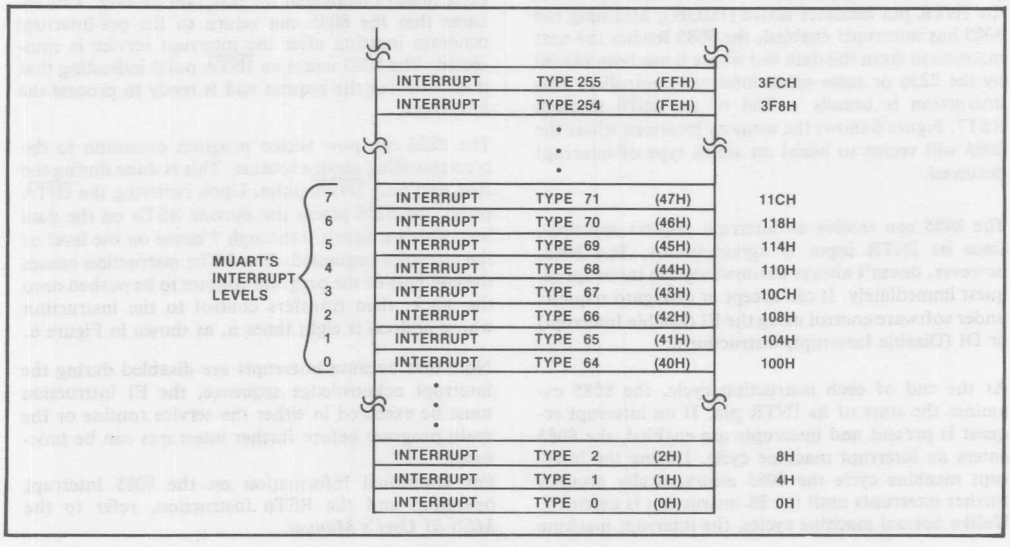


Figure 7. 8086/8088 Interrupt Vector Table

Once the service routine is completed the main program may be reentered by using an IRET (Interrupt Return) instruction. The IRET instruction will pop the pre-interrupt instruction pointer, code segment and flags off the stack. Thus the main program will resume where it was interrupted with the same flag status regardless of changes in the service routine. Note especially that this includes the state of the IF flag; thus interrupts are re-enabled automatically when returning from the service routine. For further information refer to the *iAPX 86,88 User's Manual.*

Using the 8256's Interrupt Controller Without $\overline{\text{INTA}}$

There are several configurations where the 8256 will not have an $\overline{\text{INTA}}$ signal connected to it. Some examples are when using the 8256 with an 8051 or 8048, or when connecting the INT pin on the 8256 to the 8085's RST 7.5, RST 6.5, or RST 5.5 inputs. In these configurations the IAE bit in Command Register 3 is set to 0, and the $\overline{\text{INTA}}$ pin on the 8256 is tied high. When the interrupt occurs the CPU should branch to a service routine which reads the interrupt address register to determine which interrupt request level occured. The interrupt address register contains the level of the interrupt multiplied by four. Reading the interrupt address register is equivalent in effect to the $\overline{\text{INTA}}$ signal; it clears the INT pin and indicates to the MUART that the interrupt request has been acknowledged. After the CPU reads the value in the interrupt address register, it can add an offset to this value and branch to an interrupt vector table which contains jump instructions to the appropriate interrupt service routines. An 8085 program which demonstrates this routine is given is Figure 8.

Table 3 summarizes the priority levels and the interrupt vectors which the 8256 sends back to the CPU. Note that when using Timer 1 there is a conflict present between RST0 in the 8085 mode and a hardware reset, because both expect instructions starting at address 0H. However, there is a way to distinguish between the two. After a hardware reset, all control registers are reset to a value of 0H; therefore when using Timer 1, Reset and RST0 can be distinguished by reading one of the control registers of the 8256 which has not been programmed with a value of 0H. The control registers will contain the previously programmed values if RST0 occurs.

Interrupt Registers

The 8256's interrupt controller has several registers associated with it: an Interrupt Mask Register, an Interrupt Address Register, an Interrupt Request Register, an Interrupt Service Register, and a Priority Controller. Only the Interrupt Mask Registers and the Interrupt Address Register can be accessed by the user.

Interrupt Mask Registers

The Interrupt Mask Registers consist of two write registers — the Set Interrupts Register and Reset Interrupts Register, and one read register — the Interrupt Enable Register. Each one of the eight levels of interrupts may be individually enabled or disabled through these registers. Writing a one to any of the bits in the Set Interrupts Register enables the corresponding interrupt level, while writing a one to a bit in the Reset Interrupts Register disables the corresponding interrupt level. Reading the Interrupt Enable Register allows the user to determine which interrupt levels are enabled. The bits which are set to one in the Interrupt Enable Register correspond to the levels which are enabled. All of the interrupt levels will remain enabled until disabled by the Reset Interrupts Register except the counter/timer interrupts which automatically disable themselves when they reach zero.

```
INTA:   IN      INTADD      ;Read the Interrupt Address Register
        MOV     L, A        ;Put the interrupt address in HL
        XRA     A
        MOV     H, A

        LXI     B, TABLE    ;Load BE with the interrupt table offset
        DAD     B           ;Add the offset to the interrupt address
        PCHL                ;Jump to the interrupt vecor table
```

Figure 8. Software Interrupt Acknowledge Routine

Table 3. Assignment of Interrupt Levels to Interrupt Sources

Interrupt Level	Restart Command 8085 mode	Interrupt Vector 8086 mode	Interrupt Address	Trigger Mode	Sources (Only one source can be assigned at any time)	Selection by
Highest Priority 0	RST0	40H	0H	edge	Timer 1	–
1	RST1	41H	4H	edge	Event Counter/Timer 2 or external interrupt request on Port 1 P17	Command word 1 BITI (bit 2)
2	RST2	42H	8H	level	Input EXTINT	–
3	RST3	43H	CH	edge	Event Counter/Timer 3 or cascaded event counters/ timers 3 and 5	Mode word T35 (bit 7)
4	RST4	44H	10H	edge	Serial receiver	–
5	RST5	45H	14H	edge	Serial transmitter	–
6	RST6	46H	28H	edge	Timer 4 or cascaded event counters/timers 2 and 4	Mode word T24 (bit 6)
7 Lowest Priority	RST7	47H	1CH	edge	Timer 5 or Port 2 with handshaking interrupt request	Mode word P2C2 – P2C0 (bits 2...0)

Note:
If no interrupt requests are pending and $\overline{\text{INTA}}$ cycle occurs, interrupt level 2 will be the default value vectored to the CPU.

Interrupt requests occurring when the corresponding interrupt level is disabled are lost. An interrupt will only occur if the interrupt is enabled before the interrupt request occurs.

Interrupt Address Register

The Interrupt Address Register contains an identifier for the currently requested interrupt level. The numerical value in this register is equal to the interrupt level mutliplied by four. It can be used in lieu of an $\overline{\text{INTA}}$ signal to vector the CPU to the appropriate interrupt service routine. Reading this register has the same effect as the $\overline{\text{INTA}}$ pulse: it clears the INT pin and indicates an interrupt acknowledgement to the MUART. If the Interrupt Address Register is read while no interrupts are pending, the external interrupt EXTINT will be the default value, 08H.

Interrupt Request Register

The Interrupt Request Register latches all pending interrupt requests unless they are masked off. The request is set whenever the associated event occurs.

Interrupt Service Register

In the fully nested mode of operation, every interrupt request which is granted service is entered into this register. The appropriate bit will be set whenever the interrupt is acknowledged by $\overline{\text{INTA}}$ or by reading the Interrupt Address Register. At the same time, the corresponding bit in the Interrupt Request Register is reset. The Interrupt Service Register bit remains set until the microcomputer transfers the End Of Interrupt command (EOI) to the device by writing it into Command Register 3. In the normal mode the bits in the Interrupt Service Register are never set.

Priority Controller

The priority controller selects the highest priority request in the Interrupt Request Register from up to eight requests pending. If the INTA signal is enabled and becomes active, the priority controller will cause the highest priority level in the Interrupt Request Register to be vectored back to the CPU, regardless of whether the 8256 is in the normal mode or the nested mode. In the normal mode, if any bits are set in the Interrupt Request Register, the INT pin is activated. The highest priority level in the Interrupt Request register will be transferred to the Interrupt Address Register at the same time the interrupt request occurs. In the Fully Nested mode, the priorities of all pending requests are compared to the priorities in the Interrupt Service Register. If there is a higher priority in the Interrupt Request Register than in the Interrupt Service Register, the INT signal will be activated and the new interrupt level will be loaded into the Interrupt Address Register.

Interrupt Modes

There are two modes of operation for the interrupt controller: a normal mode and a fully nested mode. In the normal mode the CPU should only be a maximum of one interrupt level deep; therefore, the CPU can be interrupted only while in the main program and not while in an interrupt service routine. In the fully nested mode it is possible for the CPU to be nested up to eight interrupt levels deep. Using the fully nested mode, the MUART will activate the INT pin only when a higher priority than the one in service is requested. The fully nested mode is used to protect high priority interrupt service routines from being interrupted by equal or lower priority requests.

Normal Mode

In the normal mode of operation the 8256 will activate the INT pin whenever any of the bits in the Interrupt Request Register are set. The bits in the Interrupt Request Register can be set only if the corresponding interrupts are enabled. If more than one interrupt request bit is set, the MUART will always place the highest priority level in the Interrupt Address Register and vector this level to the CPU during an INTA cycle. When the CPU acknowledges the interrupt request, using either the INTA signal or by reading the Interrupt Address Register, the corresponding Interrupt Request Register bit is reset. Since the Interrupt Service Register bits are never set, there is no indication in the MUART that an interrupt service routine is in progress. Therefore, the priority controller will interrupt the CPU again if any of the interrupt request bits are set, regardless of whether the next request is a higher, lower, or equal priority.

The implied way to design a program using the normal mode is to have the CPU's interrupt flag enabled during portions of the main program, but to leave the interrupt flag disabled while the CPU is executing code in an interrupt service routine. This way, the CPU can never be interrupted in an interrupt service routine. Upon completion of an interrupt service routine the program can enable the CPU's interrupt flag, then return to the main program.

Figure 9 shows an example of how the normal mode of interrupts may operate. As the CPU begins executing code in the main program, certain I/O ports, variables, and arrays need to be initialized. During this time the CPU's interrupt flag is disabled. Once the program has completed the initialization routine and can accept an interrupt, the interrupt flag is enabled. In the 8085 this is done with the assembly language instruction EI, and on the 8086 with STI.

A short time later, an interrupt request comes in on Level 4. Since the CPU's interrupt flag is enabled, the interrupt acknowledge signal is activated and the CPU branches off to Interrupt Service Routine 4. While the CPU is executing code in Interrupt Service Routine 4, an interrupt request comes in on Level 6 and then a short time later on Level 2. The 8256 activates the INT signal; however, the CPU ignores this because its interrupt flag is disabled. Upon returning to the main program the interrupt flag is enabled. When the interrupt acknowledge signal is activated, the MUART places the highest priority interrupt request on the data bus regardless of the order in which the requests came in. Therefore, during the interrupt acknowledge the MUART vectors the indirect address for Interrupt Level 2. The INT signal is not cleared after the acknowledge because there is still a pending interrupt.

The normal mode of operation is advantageous in that it simplifies programming and lowers code requirements within interrupt routines; however, there are also several disadvantages. One disadvantage is that the interrupt response time for higher priority interrupts may be excessive. For example, if the CPU is executing code in an interrupt service routine during a higher priority request, the CPU will not branch off to the higher priority service routine until the current interrupt service routine is completed. This delay time may not be acceptable for interrupts such as the serial receiver or a real time signal. For these cases the MUART provides the nested mode.

Nested Mode

In the nested mode of operation, whenever a bit in the Interrupt Request Register is set, the Priority Con-

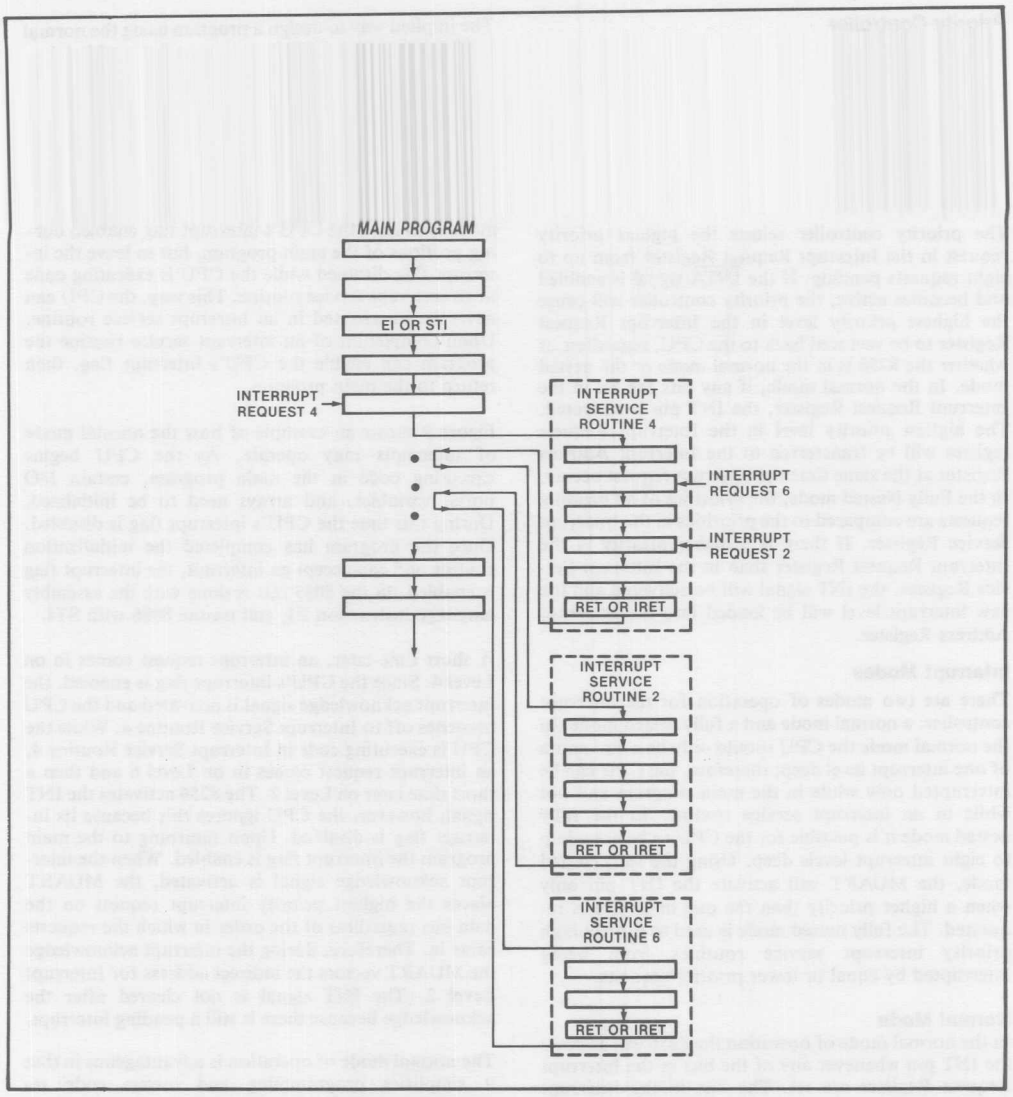


Figure 9. Normal Interrupt Mode Example

troller compares the Interrupt Request Register to the Interrupt Service Register. If the bit set in the Request Register is of a higher priority than the highest priority bit set in the Service Register, the MUART will activate the INT signal and update the Interrupt Address Register. If the bit in the Request Register is of equal or lower priority than the highest priority bit set in the Service Register, the INT signal will not be activated. When an $\overline{\text{INTA}}$ signal is activated or the Interrupt Address Register is read, the corresponding bit in the Request Register which caused the INT signal to be asserted is reset and set in the Service Register. When an EOI (End Of Interrupt) command is issued, the highest priority bit in the Service Register is reset.

Figure 10 shows an example of the program flow using the nested mode of interrupts. During the main program an interrupt request is generated from Level 4. Since the interrupt flag is enabled, the interrupt acknowledge signal is activated, and the microprocessor is vectored to Service Routine 4. During Service Routine 4, Level 2 requests an interrupt. Since Level 2 is a higher priority than Level 4, the 8256 activates its INT signal. An interrupt

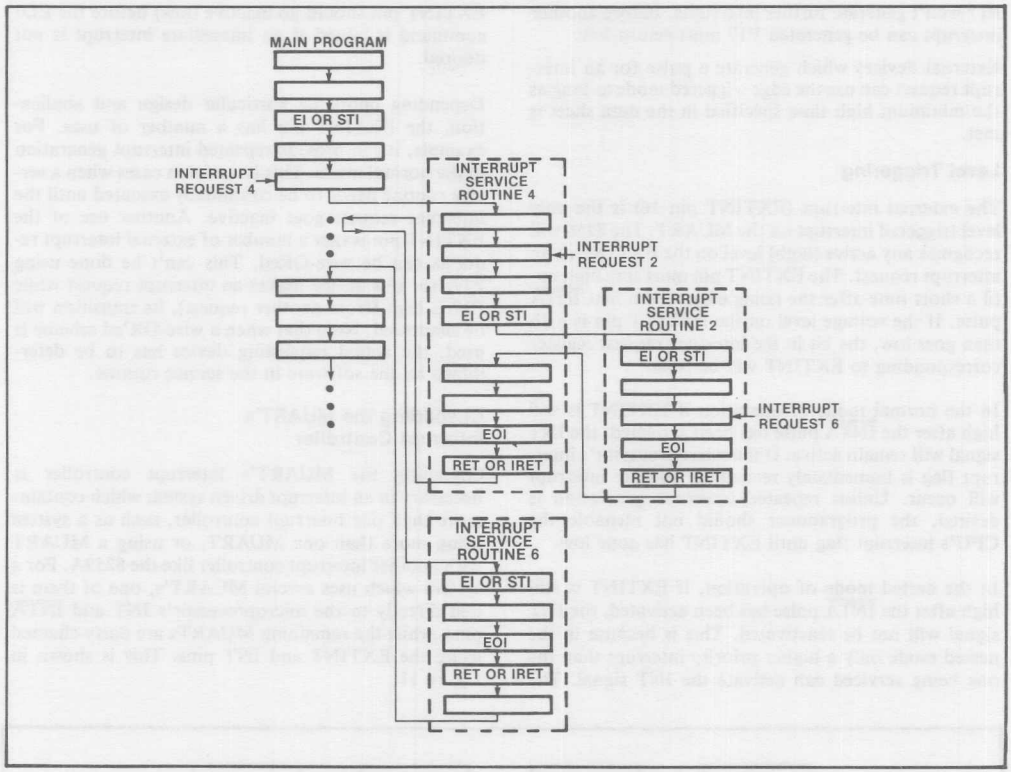


Figure 10. Fully Nested Interrupt Mode Example

acknowledge is not generated because the interrupt flag is disabled. This section of code in Service Routine 4 is protected and cannot be interrupted. A protected section of code may reinitialize a timer, take a sample, or update a global variable. When the interrupt flag is enabled the microprocessor acknowledges the interrupt and vectors into Service Routine 2. Service Routine 2 immediately enables the interrupt flag because it does not have a protected section of code. During Service Routine 2, Interrupt Request 6 is generated. However, the MUART will not interrupt the microprocessor until service routines 2 and 4 have issued the EOI command.

Edge Triggering

The MUART has a maximum of two external interrupts—EXTINT and P17. EXTINT is a dedicated interrupt pin which is level triggered, where P17 is either an I/O port or an edge triggered interrupt. If P17 is selected as an interrupt through Command Register 1 and its interrupt level is enabled, it will generate an interrupt when the level on this pin changes from low to high. The edge triggered mode incorporates an edge lockout feature. This means that after the rising edge of an interrupt request and the acknowledgment of the request, the positive level on

P17 won't generate further interrupts. Before another interrupt can be generated P17 must return low.

External devices which generate a pulse for an interrupt request can use the edge triggered mode as long as the minimum high time specified in the data sheet is met.

Level Triggering

The external interrupt (EXTINT pin 16) is the only level triggered interrupt on the MUART. The 8256 will recognize any active (high) level on the EXTINT as an interrupt request. The EXTINT pin must stay high until a short time after the rising edge of the first $\overline{INTA}$ pulse. If the voltage level on the EXTINT pin is high then goes low, the bit in the interrupt request register corresponding to EXTINT will be reset.

In the normal mode of operation if EXTINT is still high after the $\overline{INTA}$ pulse has been activated, the INT signal will remain active. If the microprocessor's interrupt flag is immediately reenabled, another interrupt will occur. Unless repeated interrupt generation is desired, the programmer should not reenable the CPU's interrupt flag until EXTINT has gone low.

In the nested mode of operation, if EXTINT is still high after the $\overline{INTA}$ pulse has been activated, the INT signal will not be reactivated. This is because in the nested mode only a higher priority interrupt than the one being serviced can activate the INT signal. The EXTINT pin should go inactive (low) before the EOI command is issued if an immediate interrupt is not desired.

Depending upon the particular design and application, the EXTINT pin has a number of uses. For example, it can provide repeated interrupt generation in the normal mode. This is useful in cases when a service routine needs to be continually executed until the interrupt request goes inactive. Another use of the EXTINT pin is that a number of external interrupt requests can be wire-ORed. This can't be done using P17, for if a device makes an interrupt request while P17 is high (from another request), its transition will be shadowed. Note that when a wire-OR'ed scheme is used, the actual requesting device has to be determined by the software in the service routine.

Cascading the MUART's Interrupt Controller

Cascading the MUART's interrupt controller is necessary in an interrupt driven system which contains more than one interrupt controller, such as a system using more than one MUART, or using a MUART with another interrupt controller like the 8259A. For a system which uses several MUART's, one of them is tied directly to the microprocessor's INT and $\overline{INTA}$ pins, while the remaining MUARTs are daisy-chained using the EXTINT and INT pins. This is shown in Figure 11.

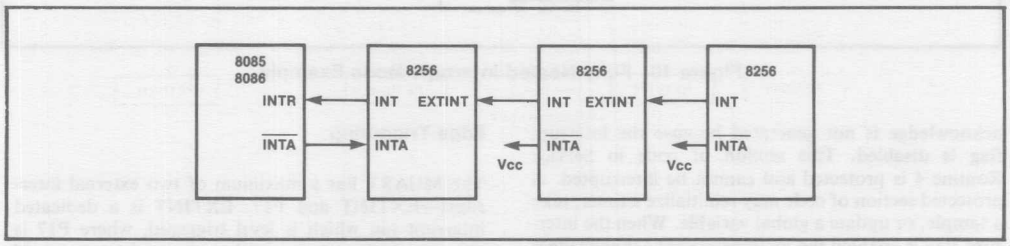


Figure 11. Cascading the MUART's Interrupt Controller

Using the configuration in Figure 11, when the microprocessor receives an interrupt, it generates an interrupt acknowledge and branches into an interrupt service routine. For the interrupt service routine of the external interrupt, EXTINT Level 2, the microprocessor will read the next MUART's interrupt address register and branch to the appropriate service routine. In effect, this would be a software interrupt acknowledge. An example of this type of interrupt acknowledge is given in Figure 8. If the last MUART in the chain indicated an external interrupt, the microprocessor would simply return to the main program; however, this would be an error condition caused by a spurious interrupt. A flow chart of the software to handle cascaded interrupts is given in Figure 12.

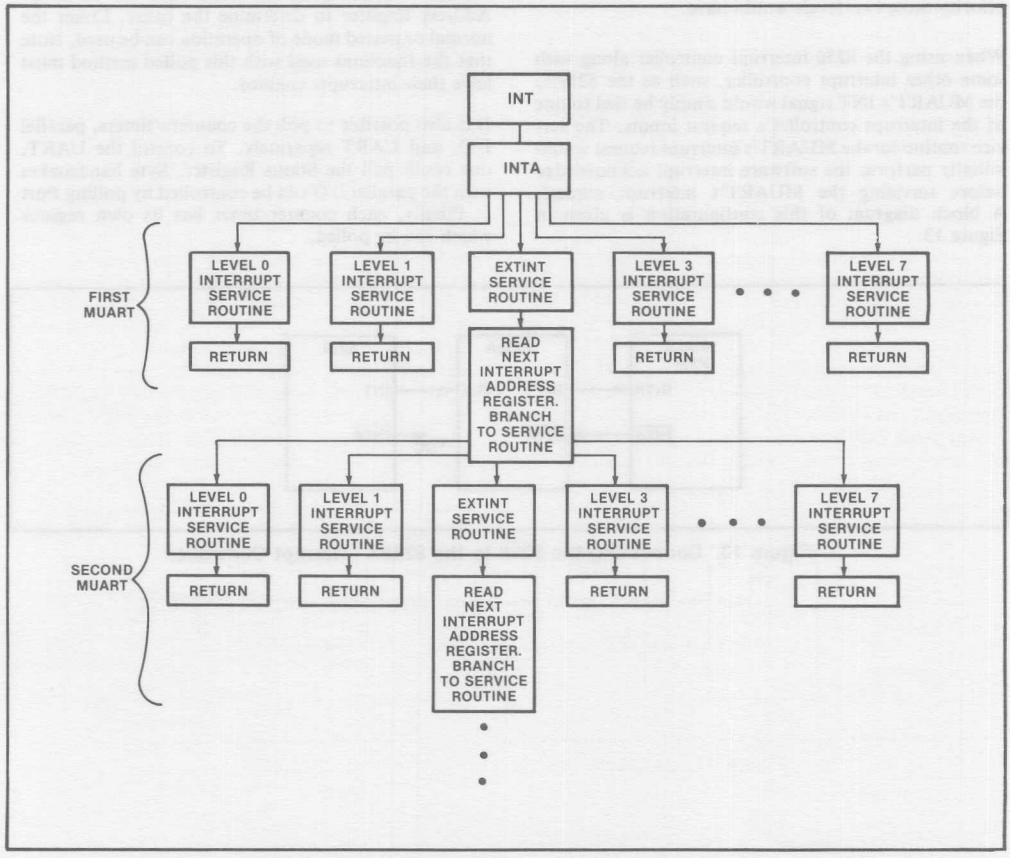


Figure 12. Flow Chart to Resolve Interrupt Request When Cascading MUART Interrupt Controllers

Some consideration should be given to the priority of the interrupts when cascading MUARTs. If all of the MUART's Level 0 and Level 1 interrupts are disabled, the highest priority interrupt is the EXTINT. In this case the last MUART in the chain would have the highest priority; however, it would take the longest time to propagate back to the CPU. If, however, Level 0 or Level 1 interrupts were enabled, the closer to the microprocessor the MUART is, the higher the priority these two levels would have.

When using the 8256 interrupt controller along with some other interrupt controller, such as the 8259A, the MUART's INT signal would simply be tied to one of the interrupt controller's request inputs. The service routine for the MUART's interrupt request would initially perform the software interrupt acknowledge before servicing the MUART's interrupt request. A block diagram of this configuration is given in Figure 13.

Polling the MUART

If interrupts are not used, the only other way to control the MUART is to poll it. It is still possible to use the priority structure of the MUART with polling. In this mode of operation the MUART's INT signal (Pin 15) is not used, and the $\overline{INTA}$ pin is tied high. Since the INT pin's level is duplicated in the MSB of the Status Register, a program can poll this bit. When it becomes set, the program could read the Interrupt Address Register to determine the cause. Either the normal or nested mode of operation can be used. Note that the functions used with this polled method must have their interrupts enabled.

It is also possible to poll the counters/timers, parallel I/O, and UART separately. To control the UART, one could poll the Status Register. Byte handshakes with the parallel I/O can be controlled by polling Port 1. Finally, each counter/timer has its own register which can be polled.

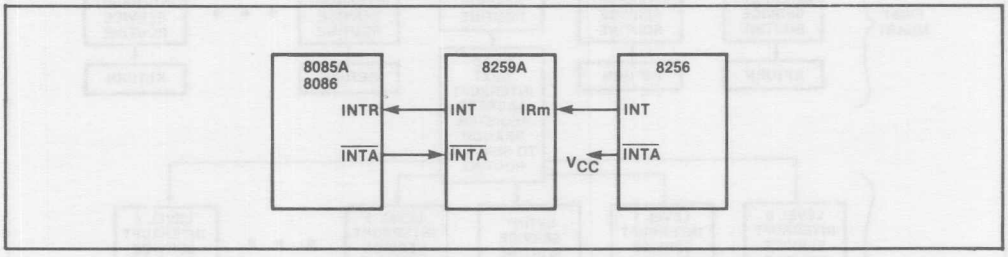


Figure 13. Connecting the 8256 to the 8259A Interrupt Controller

PIN DESCRIPTIONS

Symbol	Pin No.	Type	Name and Function
AD0-AD4 DB5-DB7	1-5 6-8	I/O	**Address/Data:** Three-state address/data lines which interface to the lower 8 bits of the microprocessor's multiplexed address/data bus. The 5-bit address is latched on the falling edge of ALE. In the 8-bit mode, AD0-AD3 are used to select the proper register, while AD1-AD4 are used in the 16-bit mode. AD4 in the 8-bit mode is ignored as an address, while AD0 in the 16-bit mode is used as a second chip select, active low.
ALE	9	I	**Address Latch Enable:** Latches the 5 address lines on AD0-AD4 and $\overline{CS}$ on the falling edge.
$\overline{RD}$	10	I	**Read Control:** When this signal is low, the selected register is gated onto the data bus.
$\overline{WR}$	11	I	**Write Control:** When this signal is low, the value on the data bus is written into the selected register.
RESET	12	I	**Reset:** An active high pulse on this pin forces the chip into its initial state. The chip remains in this state until control information is written.
$\overline{CS}$	13	I	**Chip Select:** A low on this signal enables the MUART. It is latched with the address on the falling edge of ALE, and $\overline{RD}$ and $\overline{WR}$ have no effect unless $\overline{CS}$ was latched low during the ALE cycle.

Symbol	Pin No.	Type	Name and Function
$\overline{INTA}$	14	I	**Interrupt Acknowledge:** If the MUART has been enabled to respond to interrupts, this signal informs the MUART that its interrupt request is being acknowledged by the microprocessor. During this acknowledgement the MUART puts an RSTn instruction on the data bus for the 8-bit mode or a vector for the 16-bit mode.
INT	15	0	**Interrupt Request:** A high signals the microprocessor that the MUART needs service.
EXTINT	16	I	**External Interrupt:** An external device can request interrupt service through this input. The input is level sensitive (high), therefore it must be held high until an INTA occurs or the interrupt address register is read.
CLK	17	I	**System Clock:** The reference clock for the baud rate generator and the timers.
RxC	18	I/O	**Receive Clock:** If the baud rate bits in Command Register 2 are all 0, this pin is an input which clocks serial data into the RxD pin on the rising edge of RxC. If baud rate bits in Command Register 2 are programmed from 1-0FH, this pin outputs a square wave whose rising

PIN DESCRIPTIONS (CONTINUED)

Symbol	Pin No.	Type	Name and Function
			edge indicates when the data on RxD is being sampled. This output remains high during start, stop, and parity bits.
RxD	19	I	**Receive Data:** Serial data input.
$\overline{CTS}$	21	I	**Clear To Send:** This input enables the serial transmitter. If 1, 1.5, or 2 stop bits are selected, $\overline{CTS}$ is level sensitive. As long as $\overline{CTS}$ is low, any character loaded into the transmitter buffer register will be transmitted serially. A single negative going pulse causes the transmission of a single character previously loaded into the transmitter buffer register. If a baud rate from 1-0FH is selected, $\overline{CTS}$ must be low for at least 1/32 of a bit, or it will be ignored. If the transmitter buffer is empty, this pulse will be ignored. If this pulse occurs during the transmission of a character up to the time where 1/2 of the first (or only) stop bit is sent out, it will be ignored. If it occurs afterwards, but before the end of the stop bits, the next character will be transmitted immediately following the current one. If $\overline{CTS}$ is still high when the transmitter register is sending the last stop bit, the transmitter will enter its idle state until the next high-to-low transition on $\overline{CTS}$ occurs.

Symbol	Pin No.	Type	Name and Function
			If 0.75 stop bits is chosen, the $\overline{CTS}$ input is edge sensitive. A negative edge on $\overline{CTS}$ results in the immediate transmission of the next character. The length of the stop bits is determined by the time interval between the beginning of the first stop bit and the next negative edge on $\overline{CTS}$. A high-to-low transition has no effect if the transmitter buffer is empty or if the time interval between the beginning of the stop bit and next negative edge is less than 0.75 bits. A high or a low level or a low-to-high transition has no effect on the transmitter for the 0.75 stop bit mode.
TxC	22	I/O	**Transmit Clock:** If the baud rate bits in command register 2 are all set to 0, this input clocks data out of the transmitter on the falling edge. If baud rate bits are programmed for 1 or 2, this input permits the user to provide a 32x or 64x clock which is used for the receiver and transmitter. If the baud rate bits are programmed for 3-0FH, the internal transmitter clock is output. As an output it delivers the transmitter clock at the selected bit rate. If 1½ or 0.75 stop bits are selected, the transmitter divider will be asynchronously reset at the beginning of each

PIN DESCRIPTIONS (CONTINUED)

Symbol	Pin No.	Type	Name and Function
			start bit, immediately causing a high-to-low transition on TxC. TxC makes a high-to-low transition at the beginning of each serial bit, and a low-to-high transition at the center of each bit.
TxD	23	O	**Transmit Data:** Serial data output.
P27-P20	24-31	I/O	**Parallel I/O Port 2:** Eight bit general purpose I/O port. Each nibble (4 bits) of this port can be either an input or an output. The outputs are latched whereas the input signals are not. Also, this port can be used as an 8-bit input or output port when using the two-wire handshake. In the handshake mode both inputs and outputs are latched.
P17-P10	32-39	I/O	**Parallel I/O Port 1:** Each pin can be programmed as an input or an output to perform general purpose I/O. All outputs are latched whereas inputs are not. Alternatively these pins can serve as control pins which extend the functional spectrum of the chip.
GND	20	PS	**Ground:** Power supply and logic ground reference.
Vcc	40	PS	**Power:** +5V power supply.

DESCRIPTION OF THE REGISTERS

The following section will provide a description of the registers and define the bits within the registers where appropriate. Table 4 lists the registers and their addresses.

Command Register 1

L1	L0	S1	S0	BRKI	BITI	8086	FRQ
(0R)					(0W)		

FRQ — Timer Frequency Select

This bit selects between two frequencies for the five timers. If FRQ = 0, the timer input frequency is 16KHz (62.5*us*). If FRQ = 1, the timer input frequency is 1 KHz (1ms). The selected clock frequency is shared by all the counter/timers enabled for timing; thus, all timers must run with the same time base.

8086 — 8086 Mode Enable

This bit selects between 8085 mode and 8086/8088 mode. In 8085 mode (8086 = 0), A0 to A3 are used to address the internal registers, and an RSTn instruction is generated in response to the first $\overline{\text{INTA}}$. In 8086 mode (8086 = 1), A1 to A4 are used to address the internal registers, and A0 is used as an extra chip select (A0 must equal zero to be enabled). The response to $\overline{\text{INTA}}$ is for 8086 interrupts where the first $\overline{\text{INTA}}$ is ignored, and an interrupt vector (40H to 47H) is placed on the bus in response to the second $\overline{\text{INTA}}$.

BITI — Interrupt on Bit Change

This bit selects between one of two interrupt sources on Priority Level 1, either Counter/Timer 2 or Port 1 P17 interrupt. When this bit equals 0, Counter/Timer 2 will be mapped into Priority Level 1. If BITI equals 0 and Level 1 interrupt is enabled, a transition from 1 to 0 in Counter/Timer 2 will generate an interrupt request on Level 1. When BITI equals 1, Port 1 P17 external edge triggered interrupt source is mapped into Priority Level 1. In this case if Level 1 is enabled, a low-to-high transition on P17 generates an interrupt request on Level 1.

BRKI — Break-In Detect Enable

If this bit equals 0, Port 1 P16 is a general purpose I/O port. When BRKI equals 1, the Break-In Detect feature is enabled on Port 1 P16. A Break-In condition is present on the transmission line when it is forced to the start bit voltage level by the receiving station. Port 1 P16 must be connected externally to the transmission line in order to detect a Break-In. A

Table 4. MUART Registers

Read Registers								8085 Mode: AD3 AD2 AD1 AD0 8086 Mode: AD4 AD3 AD2 AD1				Write Registers							
L1	L0	S1	S0	BRKI	BITI	8086	FRQ	0	0	0	0	L1	L0	S1	S0	BRKI	BITI	8086	FRQ
Command 1												Command 1							
PEN	EP	C1	C0	B3	B2	B1	B0	0	0	0	1	PEN	EP	C1	C0	B3	B2	B1	B0
Command 2												Command 2							
0	RxE	IAE	NIE	0	SBRK	TBRK	0	0	0	1	0	SET	RxE	IAE	NIE	END	SBRK	TBRK	RST
Command 3												Command 3							
T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0	0	0	1	1	T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
Mode												Mode							
P17	P16	P15	P14	P13	P12	P11	P10	0	1	0	0	P17	P16	P15	P14	P13	P12	P11	P10
Port 1 Control												Port 1 Control							
L7	L6	L5	L4	L3	L2	L1	L0	0	1	0	1	L7	L6	L5	L4	L3	L2	L1	L0
Interrupt Enable												Set Interrupts							
D7	D6	D5	D4	D3	D2	D1	D0	0	1	1	0	L7	L6	L5	L4	L3	L2	L1	L0
Interrupt Address												Reset Interrupts							
D7	D6	D5	D4	D3	D2	D1	D0	0	1	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Receiver Buffer												Transmitter Buffer							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	0	0	D7	D6	D5	D4	D3	D2	D1	D0
Port 1												Port 1							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Port 2												Port 2							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 1												Timer 1							
D7	D6	D5	D4	D3	D2	D1	D0	1	0	1	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 2												Timer 2							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	0	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 3												Timer 3							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	0	1	D7	D6	D5	D4	D3	D2	D1	D0
Timer 4												Timer 4							
D7	D6	D5	D4	D3	D2	D1	D0	1	1	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Timer 5												Timer 5							
INT	RBF	TBE	TRE	BD	PE	OE	FE	1	1	1	1	0	RS4	RS3	RS2	RS1	RS0	TME	DSC
Status												Modification							

Break-In is polled by the MUART during the transmission of the last or only stop bit of a character.

A Break-In Detect is OR-ed with Break Detect in Bit 3 of the Status Register. The distinction can be made through the interrupt controller. If the transmit and receive interrupts are enabled, a Break-In will generate an interrupt on Level 5, the transmit interrupt, while Break will generate an interrupt on Level 4, the receive interrupt.

S0, S1 — Stop Bit Length

S1	S0	Stop Bit Length
0	0	1
0	1	1.5
1	0	2
1	1	0.75

The relationship of the number of stop bits and the function of input $\overline{CTS}$ is discussed in the Pin Description section under "$\overline{CTS}$".

L0, L1 — Character Length

L1	L0	Character Length
0	0	8
0	1	7
1	0	6
1	1	5

Command Register 2

PEN	EP	C1	C0	B3	B2	B1	B0
	(1R)				(1W)		

Programming bits 0...3 with values from 3H to FH enables the internal baud rate generator as a common clock source for the transmitter and receiver and determines its divider ratio.

Programming bits 0...3 with values of 1H or 2H enables input TxC as a common clock source for the transmitter and receiver. The external clock must provide a frequency of either 32x or 64x the baud rate. The data transmission rates range from 0...32 Kbaud.

If bits 0...3 are set to 0, separate clocks must be input to pin RxC for the receiver and pin TxC for the transmitter. Thus, different baud rates can be used for transmission and reception. In this case, prescalers are disabled and the input serial clock frequency must match the baud rate. The input serial clock frequency can range from 0 to 1.024 MHz.

B0, B1, B2, B3 — Baud Rate Select

These four bits select the bit clock's source, sampling rate, and serial bit rate for the internal baud rate generator.

B3	B2	B1	B0	Baud Rate	Sampling Rate
0	0	0	0	$\overline{TxC}$, $\overline{RxC}$	1
0	0	0	1	$\overline{TxC}$/64	64
0	0	1	0	$\overline{TxC}$/32	32
0	0	1	1	19200	32
0	1	0	0	9600	64
0	1	0	1	4800	64
0	1	1	0	2400	64
0	1	1	1	1200	64
1	0	0	0	600	64
1	0	0	1	300	64
1	0	1	0	200	64
1	0	1	1	150	64
1	1	0	0	110	64
1	1	0	1	100	64
1	1	1	0	75	64
1	1	1	1	50	64

The following table gives an overview of the function of pins TxC and RxC:

Bits 3 to 0 (Hex.)	TxC	RxC
0	Input: 1 x baud rate clock for the transmitter	Input: 1 x baud rate clock for the receiver
1, 2	Input 32 x or 64 x baud rate for transmitter and receiver	Output: receiver bit clock with a low-to-high transition at data bit sampling time. Otherwise: high level
3 to F	Output: baud rate clock of the transmitter	Output: as above

As an output, RxC outputs a low-to-high transition at sampling time of evary data bit of a character. Thus, data can be loaded, e.g., into a shift register external-

ly. The transition occurs only if data bits of a character are present. It does not occur for start, parity, and stop bits (RxC = high).

As an output, TxC outputs the internal baud rate clock of the transmitter. There will be a high-to-low transition at every beginning of a bit.

C0, C1 — System Clock Prescaler (Bits 4, 5)

Bits 4 and 5 define the system clock prescaler divider ratio. The internal operating frequency of 1.024 MHz is derived from the system clock.

C1	C0	Divider Ratio	Clock at Pin CLK
0	0	5	5.12 MHz
0	1	3	3.072 MHz
1	0	2	2.048 MHz
1	1	1	1.024 MHz

EP — Even Parity (Bit 6)

EP = 0: Odd parity
EP = 1: Even parity

PEN — Parity Enable (Bit 7)

Bit 7 enables parity generation and checking.

PEN = 0: No parity bit
PEN = 1: Enable parity bit

The parity bit according to Command Register 2 bit 6 (see above) is inserted between the last data bit of a character and the first or only stop bit. The parity bit is checked during reception. A false parity bit generates an error indication in the Status Register and an Interrupt Request on Level 4.

Command Register 3

SET	RxE	IAE	NIE	END	SBRK	TBRK	RST
(2R)					(2W)		

Command Register 3 is different from the first two registers because it has a bit set/reset capability. Writing a byte with Bit 7 high sets any bits which were also high. Writing a byte with Bit 7 low resets any bits which were high. If any bit 0-6 is low, no change occurs to that bit. When Command Register 3 is read, bits 0, 3, and 7 will always be zero.

RST — Reset

If RST is set, the following events occur:

1) All bits in the Status Register except bits 4 and 5 are cleared, and bits 4 and 5 are set.

2) The Interrupt Enable, Interrupt Request, and Interrupt Service Registers are cleared. Pending requests and indications for interrupts in service will be cancelled. Interrupt signal INT will go low.

3) The receiver and transmitter are reset. The transmitter goes idle (TxD is high), and the receiver enters start bit search mode.

4) If Port 2 is programmed for handshake mode, IBF and OBF are reset high.

RST does *not* alter ports, data registers or command registers, but it halts any operation in progress. RST is automatically cleared.

RST = 0 has no effect. The reset operation triggered by Command Register 3 is a subset of the hardware reset.

TBRK — Transmit Break

The transmission data output TxD will be set low as soon as the transmission of the previous character has been finished. It stays low until TBRK is cleared. The state of $\overline{CTS}$ is of no significance for this operation. As long as break is active, data transfer from the Transmitter Buffer to the Transmitter Register will be inhibited. As soon as TBRK is reset, the break condition will be deactivated and the transmitter will be reenabled.

SBRK — Single Character Break

This causes the transmitter data to be set low for one character including start bit, data bits, parity bit, and stop bits. SBRK is automatically cleared when time for the last data bit has passed. It will start after the character in progress completes, and will delay the next data transfer from the Transmitter Buffer to the Transmitter Register until TxD returns to an idle

(marking) state. If both TBRK and SBRK are set, break will be set as long as TBRK is set, but SBRK will be cleared after one character time of break. If SBRK is set again, it remains set for another character. The user can send a definite number of break characters in this manner by clearing TBRK after setting SBRK for the last character time.

END — End of Interrupt

If fully nested interrupt mode is selected, this bit resets the currently served interrupt level in the Interrupt Service Register. *This command must occur at the end of each interrupt service routine during fully nested interrupt mode.* END is automatically cleared when the Interrupt Service Register (internal) is cleared. END is ignored if nested interrupts are not enabled.

NIE — Nested Interrupt Enable

When NIE equals 1, the interrupt controller will operate in the nested interrupt mode. When NIE equals 0, the interrupt controller will operate in the normal interrupt mode. Refer to the "Interrupt controller" section under "Normal Mode" and "Nested Mode" for a detailed description of these operations.

IAE — Interrupt Acknowledge Enable

This bit enables an automatic response to $\overline{\text{INTA}}$. The particular response is determined by the 8086 bit in Command Register 1.

RxE — Receive Enable

This bit enables the serial receiver and its associated status bits in the status register. If this bit is reset, the serial receiver will be disabled and the receive status bits will not be updated.

Note that the detection of break characters remains enabled while the receiver is disabled; i.e., Status Register Bit 3 (BD) will be set while the receiver is disabled whenever a break character has been recognized at the receive data input RxD.

SET — Bit Set/Reset

If this bit is high during a write to Command Register 3, then any bit marked by a high will set. If this bit is low, then any bit marked by a high will be cleared.

Mode Register

T35	T24	T5C	CT3	CT2	P2C2	P2C1	P2C0
(3R)					(3W)		

P2C2, P2C1, P2C0 — Port 2 Control

P2C2	P2C1	P2C0	Mode	Direction Upper	Direction Lower
0	0	0	nibble	input	input
0	0	1	nibble	input	output
0	1	0	nibble	output	input
0	1	1	nibble	output	output
1	0	0	byte handshake	input	
1	0	1	byte handshake	output	
1	1	0	*DO NOT USE*		
1	1	1	test		

If test mode is selected, the output from the internal baud rate generator is placed on bit 4 of Port 1 (pin 35).

To achieve this, it is necessary to program bit 4 of Port 1 as an output (Port 1 Control Register Bit P14 = 1), and to program Command Register 2 bits B3 - B0 with a value ≥ 3H.

Note:
If Port 2 is operating in handshake mode, Interrupt Level 7 is not available for Timer 5. Instead it is assigned to Port 2 handshaking.

CT2, CT3 — Counter/Timer Mode

Bit 3 and 4 defines the mode of operation of event counter/timers 2 and 3 regardless of its use as a single unit or as a cascaded one.

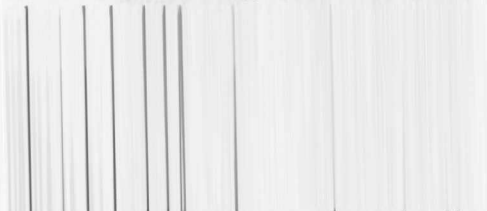
If CT2 or CT3 are high, then counter/timer 2 or 3 respectively is configured as an event counter on bit 2 or 3 respectively of Port 1 (pins 37 or 36). The event counter decrements the count by one on each low-to-high transition of the external input. If CT2 or CT3 is low, then the respective counter/timer is configured as a timer and the Port 1 pins are used for parallel I/O.

T5C — Timer 5 Control

If T5C is set, then Timer 5 can be preset and started by an external signal. Writing to the Timer 5 register loads the Timer 5 save register and stops the timer. A high-to-low transition on bit 5 of Port 1 (pin 34) loads the timer with the saved value and starts the timer. The next high-to-low transition on pin 34 retriggers the timer by reloading it with the initial value and continues timing.

Following a hardware reset, the save register is reset to 00H and both clock and trigger inputs are disabled. Transferring an instruction with T5C = 1 enables the trigger input; the save register can now be loaded with

an initial value. The first trigger pulse causes the initial



value to be loaded from the save register and enables the counter to count down to zero.

When the timer reaches zero it issues an interrupt request, disables its interrupt level and continues counting. A subsequent high-to-low transition on pin 5 resets Timer 5 to its initial value. For another timer interrupt, the Timer 5 interrupt enable bit must be set again.

T35, T24 — Cascade Timers

These two bits cascade Timers 3 and 5 or 2 and 4. Timers 2 and 3 are the lower bytes, while Timers 4 and 5 are the upper bytes. If T5C is set, then both Timers 3 and 5 can be preset and started by an external pulse. When a high-to-low transition occurs, Timer 5 is

preset to its saved value, But Timer 3 is always preset to all ones. If either CT2 or CT3 is set, then the corresponding timer pair is a 16-bit event counter.

A summary of the counter/timer control bits is given in Table 5.

Note:
Interrupt levels assigned to single counters are partly not occupied if event counters/timers are cascaded. Level 2 will be vacated if event counters/timers 2 and 4 are cascaded. Likewise, Level 7 will be vacated if event counters/timers 3 and 5 are cascaded.

Single event counters/timers generate an interrupt request on the transition from 01H to 00H, while cascaded ones generate it on the transition from 0001H to 0000H.

Table 5. Event Counters/Timers Mode of Operation

Event Counter/ Timer	Function	Programming (Mode Word)	Clock Source
1	8-bit timer	–	internal clock
2	8-bit timer	T24 = 0, CT2 = 0	internal clock
	8-bit event counter	T24 = 0, CT2 = 1	P12 pin 37
3	8-bit timer	T35 = 0, CT3 = 0	internal clock
	8-bit event counter	T35 = 0, CT3 = 1	P13 pin 36
4	8-bit timer	T24 = 0	internal clock
5	8-bit timer, normal mode	T35 = 0, T5C = 0	internal clock
	8-bit timer, retriggerable mode	T35 = 0, T5C = 1	internal clock
2 and 4 cascaded	16-bit timer	T24 = 1, CT2 = 0	internal clock
	16-bit event counter	T24 = 1, CT2 = 1	P12 pin 37
3 and 5 cascaded	16-bit timer, normal mode	T35 = 1, T5C = 0, CT3 = 0	internal clock
	16-bit event counter, normal mode	T35 = 1, T5C = 0, CT3 = 1	P13 pin 36
	16-bit timer, Retriggerable mode	T35 = 1, T5C = 1, CT3 = 0	internal clock
	16-bit event counter, Retriggerable mode	T35 = 1, T5C = 1, CT3 = 1	P13 pin 36

Port 1 Control Register

P17	P16	P15	P14	P13	P12	P11	P10
(4R)					(4W)		

Each bit in the Port 1 Control Register configures the direction of the corresponding pin. If the bit is high, the pin is an output, and if it is low the pin is an input. Every Port 1 pin has another function which is controlled by other registers. If that special function is disabled, the pin functions as a general I/O pin as specified by this register. The special functions for each pin are described below.

Port 10, 11 — Handshake Control

If byte handshake control is enabled for Port 2 by the Mode Register, then Port 10 is programmed as $\overline{STB}/\overline{ACK}$ handshake control input, and Port 11 is programmed as $\overline{IBF}/\overline{OBF}$ handshake control output.

If byte handshake mode is enabled for output on Port 2, $\overline{OBF}$ indicates that a character has been loaded into the Port 2 output buffer. When an external device reads the data, it acknowledges this operation by driving $\overline{ACK}$ low. $\overline{OBF}$ is set low by writing to Port 2 and is reset high by $\overline{ACK}$.

If byte handshake mode is enabled for input on Port 2, $\overline{STB}$ is an input. $\overline{IBF}$ is driven low after $\overline{STB}$ goes low. On the rising edge of $\overline{STB}$ the data from Port 2 is latched.

$\overline{IBF}$ is reset high when Port 2 is read.

Port 12, 13 — Counter 2, 3 Input

If Timer 2 or Timer 3 is programmed as an event counter by the Mode Register, then Port 12 or Port 13 is the counter input for Event Counter 2 or 3, respectively.

Port 14 — Baud Rate Generator Output Clock

If test mode is enabled by the Mode Register and Command Register 2 baud rate select is greater than 2, then Port 14 is an output from the internal baud rate generator.

P14 in Port 1 control register must be set to 1 for the baud rate generator clock to be output. The baud rate generator clock is 64 x the serial bit rate except at 19.2Kbps when it is 32 x the bit rate.

Port 15 — Timer 5 Trigger

If T5C is set in the Mode Register enabling a retriggerable timer, then Port 15 is the input which starts and reloads Timer 5.

A high-to-low transition on P15 (Pin 34) loads the timer with the save register and starts the timer.

Port 16 — Break-In Detect

If Break-In Detect is enabled by BRKI in Command Register 1, then this input is used to sense a Break-In. If Port 16 is low while the serial transmitter is sending the last stop bit, then a Break-In condition is signaled.

Port 17 — Port Interrupt Source

If BITI in Command Register 1 is set, then a low-to-high transition on Port 17 generates an interrupt request on Priority Level 1.

Port 17 is edge triggered.

Interrupt Enable Register

L7	L6	L5	L4	L3	L2	L1	L0
		(5R)		(5W = enable, 6W = disable)			

Interrupts are enabled by writing to the Set Interrupts Register (5W). Interrupts are disabled by writing to the Reset Interrupts Register (6W). Each bit set by the Set Interrupts Register (5W) will enable that level interrupt, and each bit set in the Reset Interrupts Register (6W) will disable that level interrupt. The user can determine which interrupts are enabled by reading the Interrupt Enable Register (5R).

Priority		Source
Highest	L0	Timer 1
	L1	Timer 2 or Port Interrupt
	L2	External Interrupt (EXTINT)
	L3	Timer 3 or Timers 3 & 5
	L4	Receiver Interrupt
	L5	Transmitter Interrupt
	L6	Timer 4 or Timers 2 & 4
Lowest	L7	Timer 5 or Port 2 Handshaking

Interrupt Address Register

0	0	0	D4	D3	D2	0	0
(6R)					Interrupt Level Indication		

Reading the interrupt address register transfers an identifier for the currently requested interrupt level on the system data bus. This identifier is the number of the interrupt level multiplied by 4. It can be used by the CPU as an offset address for interrupt handling. Reading the interrupt address register has the same effect as a hardware interrupt acknowledge $\overline{INTA}$; it clears the interrupt request pin (INT) and indicates an interrupt acknowledgement to the interrupt controller.

Receiver and Transmitter Buffer

D7	D6	D5	D4	D3	D2	D1	D0
	(7R)				(7W)		

Both the receiver and transmitter in the MUART are double buffered. This means that the transmitter and receiver have a shift register and a buffer register. The buffer registers are directly addressable by reading or writing to register seven. After the receiver buffer is full, the RBF bit in the status register is set. Reading the receive buffer clears the RBF status bit. The transmit buffer should be written to only if the TBE bit in the status register is set. Bytes written to the transmit buffer are held there until the transmit shift register is empty, assuming $\overline{CTS}$ is low. If the transmit buffer and shift register are empty, writing to the transmit buffer immediately transfers the byte to the transmit shift register. If a serial character length is less than 8 bits, the unused most significant bits are set to zero when reading the receive buffer, and are ignored when writing to the transmit buffer.

Port 1

D7	D6	D5	D4	D3	D2	D1	D0
	(8R)				(8W)		

Writing to Port 1 sets the data in the Port 1 output latch. Writing to an input pin does not affect the pin, but the data is stored and will be output if the direction of the pin is changed later. If the pin is used as a control signal, the pin will not be affected, but the data is stored. Reading Port 1 transfers the data in Port 1 onto the data bus.

Port 2

D7	D6	D5	D4	D3	D2	D1	D0
	(9R)				(9W)		

Writing to Port 2 sets the data in the Port 2 output latch. Writing to an input pin does not affect the pin, but it does store the data in the latch. Reading Port 2 puts the input pins onto the bus or the contents of the output latch for output pins.

Timer 1-5

D7	D6	D5	D4	D3	D2	D1	D0
	($0A_{16}$-$0E_{16}$R)				($0A_{16}$-$0E_{16}$W)		

Reading Timer N puts the contents of the timer onto the data bus. If the counter changes while $\overline{RD}$ is low, the value on the data bus will not change. If two timers are cascaded, reading the high-order byte will cause the low-order byte to be latched. Reading the low-order byte will unlatch them both. Writing to either timer or decascading them also clears the latch condition. Writing to a timer sets the starting value of that timer. If two timers are cascaded. writing to the high-order byte presets the low-order byte to all ones. Loading only the high-order byte with a value of X leads to a count of X 256 + 255. Timers count down continuously. If the interrupt is enabled, it occurs when the counter changes from 1 to 0.

The timer/counter interrupts are automatically disabled when the interrupt request is generated.

Status Register

INT	RBF	TBE	TRE	BD	PE	OE	FE
			($0F_{16}$R)				

Reading the status register gates its contents onto the data bus. It holds the operational status of the serial interface as well as the status of the interrupt pin INT. The status register can be read at any time. The flags are stable and well defined at all instants.

FE — Framing Error, Transmission Mode

Bit 0 can be used in two modes. Normally, FE indicates framing error which can be changed to transmission mode indication by setting the TME bit in the modification register.

If transmission mode is disabled (in Modification Register), then FE indicates a framing error. A framing error is detected during the *first* stop bit. The error is reset by reading the Status Register or by a chip reset. A framing error does not inhibit the loading of the Receiver Buffer. If RxD remains low, the receiver will assemble the next character. The false stop bit is treated as the next start bit, and no high-to-low transition on RxD is requied to synchronize the receiver.

When the TME bit in the Modification Register is set, FE is used to indicate that the transmitter was active during the reception of a character, thus indicating that the character received was transmitted by its own transmitter. FE is reset when the transmitter is not active during the reception of character. Reading the status register will not reset the FE bit in the transmission mode.

OE — Overrun Error

If the user does not read the character in the Receiver Buffer before the next character is received and transferred to this register, then the OE bit is set. The OE flag is set during the reception of the first stop bit and is cleared when the Status Register is read or when a hardware or software reset occurs. The first character received in this case will be lost.

PE — Parity Error

This bit indicates that a parity error has occurred during the reception of a character. A parity error is present if value of the parity bit in the received character is different from the one expected according to command word 2 bits 6 EP. The parity bit is expected and checked only if it is enabled by command word 2 bit 7 PEN.

A parity error is set during the first stop bit and is reset by reading the Status Register or by a chip reset.

BD — Break/Break-In

The BD bit flags whether a break character has been received, or a Break-In condition exists on the transmission line. Command Register 1 Bit 3 (BRKI) enables the Break-In Detect function.

Whenever a break character has been received, Status Register Bit 3 will be set and in addition an interrupt request on Level 4 is generated. The receiver will be idled. It will be started again with the next high-to-low transition at pin RxD.

The break character received will not be loaded into the receiver buffer register.

If Break-In Detection is enabled and a Break-In condition occurs, Status Register Bit 3 will be set and in addition an interrupt request on Level 5 is generated.

The BD status bit will be reset on reading the status register or on a hardware or software reset. For more information on Break/Break-In, refer to the "Serial Asynchronous Communication" section under "Receive Break Detect"and "Break-In Detect."

TRE — Transmit Register Empty

When TRE is set the transmit register is empty and an interrupt request is generated on Level 5 if enabled. When TRE equals 0 the transmit register is in the process of sending data. TRE is set by a chip reset and when the last stop bit has left the transmitter. It is reset when a character is loaded into the Transmitter Register. If $\overline{CTS}$ is low, the Transmitter Register will be loaded during the transmission of the start bit. If $\overline{CTS}$ is high at the end of a character, TRE will remain high and no character will be loaded into the Transmitter Register until $\overline{CTS}$ goes low. If the transmitter was inactive before a character is loaded into the Transmitter Buffer, the Transmitter Register will be empty temporarily while the buffer is full. However, the data in the buffer will be transferred to the transmitter register immediately and TRE will be cleared while TBE is set.

TBE — Transmitter Buffer Empty

TBE indicates the Transmitter Buffer is empty and is ready to accept a character. TBE is set by a chip reset or the transfer of data to the Transmitter Register, and is cleared when a character is written to the transmitter buffer. When TBE is set, an interrupt request is generated on Level 5 if enabled.

RBF — Receiver Buffer Full

RBF is set when the Receiver Buffer has been loaded with a new character during the sampling of the first stop bit. RBF is cleared by reading the receiver buffer or by a chip reset.

INT — Interrupt Pending

The INT bit reflects the state of the INT Pin (Pin 15) and indicates an interrupt is pending. It is reset by $\overline{INTA}$ or by reading the Interrupt Address Register if only one interrupt is pending and by a chip reset.

FE, OE, PE, RBF, and Break Detect all generate a Level 4 interrupt when the receiver samples the first stop bit. TRE, TBE, and Break-In Detect generate a Level 5 interrupt. TRE generates an interrupt when TBE is set and the Transmitter Register finished transmitting. The Break-In Detect interrupt is issued at the same time as TBE or TRE.

Modification Register

0	RS4	RS3	RS2	RS1	RS0	TME	DSC

($0F_{16}W$)

DSC — Disable Start Bit Check

DSC disables the receiver's start bit check. In this state the receiver will not be reset if RxD is not low at the center of the start bit.

TME — Transmission Mode Enable

TME enables transmission mode and disables framing error detection. For information on transmission mode see the description of the framing error bit in the Status Register.

RS0, RS1, RS2, RS3, RS4 — Receiver Sample Time

The number in RSn alters when the receiver samples RxD. The receiver sample time can be modified only if the receiver is *not* clocked by RxC.

Note:
The modification register cannot be read. Reading from address 0FH, 8086: 1EH gates the contents of the status register onto the data bus.

- A hardware reset (reset, Pin 12) resets all modification register bits to 0, i.e.:
  * The start bit check is enabled.
  * Status Register Bit 0 (FE) indicates framing error.
  * The sampling time of the serial receiver is the bit center.

  A software reset (Command Word 3, RST) does not affect the modification register.

Hardware Reset

A reset signal on pin RESET (HIGH level) forces the device 8256 into a well-defined initial state. This state is characterized as follows:

RS4	RS3	RS2	RS1	RS0	Point of time between start of bit and end of bit measured in steps of 1/32 bit length	
0	1	1	1	1	1	(Start of Bit)
0	1	1	1	0	2	
0	1	1	0	1	3	
0	1	1	0	0	4	
0	1	0	1	1	5	
0	1	0	1	0	6	
0	1	0	0	1	7	
0	1	0	0	0	8	
0	0	1	1	1	9	
0	0	1	1	0	10	
0	0	1	0	1	11	
0	0	1	0	0	12	
0	0	0	1	1	13	
0	0	0	1	0	14	
0	0	0	0	1	15	
0	0	0	0	0	16	(Bit center)
1	1	1	1	1	17	
1	1	1	1	0	18	
1	1	1	0	1	19	
1	1	1	0	0	20	
1	1	0	1	1	21	
1	1	0	1	0	22	
1	1	0	0	1	23	
1	1	0	0	0	24	
1	0	1	1	1	25	
1	0	1	1	0	26	
1	0	1	0	1	27	
1	0	1	0	0	28	
1	0	0	1	1	29	
1	0	0	1	0	30	
1	0	0	0	1	31	
1	0	0	0	0	32	(End of Bit)

1) Command registers 1, 2 and 3, mode register, Port 1 control register, and modification register are reset. Thus, all bits of the parallel interface are set to be inputs and event counters/timers are configured as independent 8-bit timers.

2) Status register bits are reset with the exception of bits 4 and 5. Bits 4 and 5 are set indicating that both transmitter register and transmitter buffer register are empty.

3) The interrupt mask, interrupt request, and interrupt service register bits are reset and disable all requests. As a consequence, interrupt signal INT is inactive (LOW).

4) The transmit data output is set to the marking state (HIGH) and the receiver section is disabled until it is enabled by Command Register 3 Bit 6.

5) The start bit will be checked at sampling time. The receiver will return to start bit search mode if input RxD is not LOW at this time.

6) Status Register Bit 0 implies framing error.

7) The receiver samples input RxD at bit center.

Reset has no effect on the contents of receiver buffer register, transmitter buffer register, the intermediate latches of parallel ports, and event counters/timers, respectively.

INTERFACING

This section describes the hardware interface between the 8256 MUART and the 8085, 8086, 8088, and 80186 microprocesors. Figures 14 through 19 display the block diagrams for these interfaces. The MUART can be interfaced to many other microprocessors using these basic principles.

In all cases the 8256 will be connected directly to the CPU's multiplexed address/data bus. If latches or data bus buffers are used in a system, the MUART should be on the microprocessor side of the address/data bus. The MUART latches the address internally on the falling edge of ALE. The address consists of Chip Select (CS) and four address lines. For 8-bit microprocessors, AD0-AD3 are the address lines. For 16-bit microprocessors, AD1-AD4 are the address lines; AD0 is used as a second chip select which is active low. Since chip select is internally latched along with the address, it does not have to remain active during the entire instruction cycle. As long as the chip select setup and hold times are met, it can be derived from multiplexed address/data lines or multiplexed address/status lines.

In Figure 15, the 8088 min mode, the 8205 chip select decoder is connected to the 8088's address bus lines A8-A15. These address lines are stable throughout the entire instruction cycle. However, the MUART's chip select signal could have been derived from A16/S3-A19/S6.

Figure 16 shows the 8256 interfaced with an 8086 in the min mode. When the 8256 is in the 16-bit mode, A0 serves as a second chip select. As a result the MUART's internal registers will all have even addresses since A0 must be zero to select the device. Normally the MUART will be placed on the lower data byte. If the MUART is placed on the upper data byte the internal registers will be 512 address locations apart and the chip would occupy an 8 K word address space. Figure 16A shows a table and a diagram of how the 8256 may be selected in an 8086 system where the MUART is I/O mapped and used on the lower byte of the address/data bus.

PROGRAMMING

Initialization

In general the MUART's functions are independent of each other and only the registers and bits associated with a particular function need to be initialized, not the entire chip. The command sequence is arbitrary since every register is directly addressable; however, Command Word 1 must be loaded first. To put the device into a fully operational condition, it is necessary to write the following commands:

Command byte 1
Command byte 2
Command byte 3
Mode byte
Port 1 control
Set Interrupts

The modification register may be loaded if required for special applications; normally this operation is not necessary. It is a good idea to reset the part before initialization. (Either a hardware or a software reset will do.)

Operating the Serial Interface

The microprocessor transfers data to the serial interface by writing bytes to the Transmit Buffer Register. Receive characters are transferred by reading the Receiver Buffer Register. The Status Register provides all of the necessary information to operate the serial I/O, including when to write to the Transmit Buffer, and when to read the Receive Buffer and error information.

Transmitting

The transmitter and the receiver may be operated by using either polling or interrupts. If polling is used then the software may poll the Status Register and write a byte to the Transmit Buffer whenever TBE = 1. Writing a byte to the Transmit Buffer clears the TBE

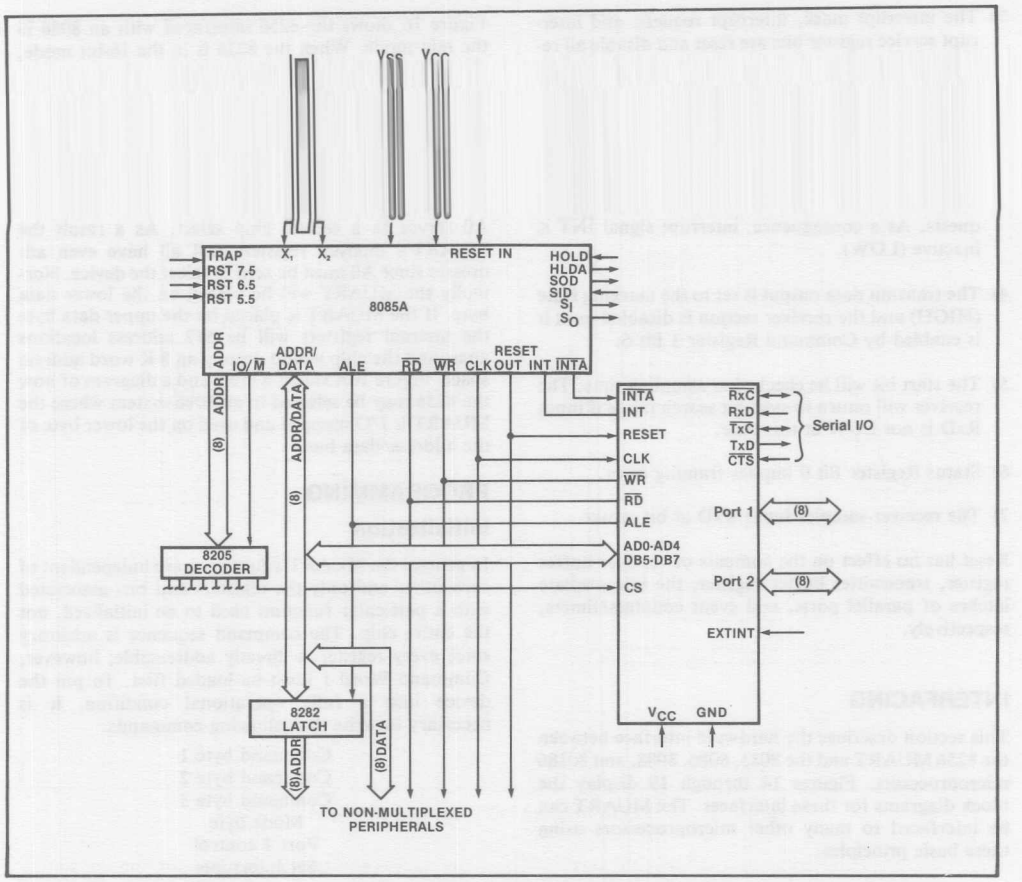


Figure 14. 8085/8256 Interface

status bit. If the $\overline{CTS}$ pin is low, then the Transmit Buffer will transfer the data to the Transmit Register when it becomes empty. When this transfer takes place the TRE bit is reset, and the TBE bit is set indicating the next byte may be written to the Transmit Buffer. If $\overline{CTS}$ is high, disabling the transmitter, the data byte will remain in the Transmit Buffer and TBE will remain low until $\overline{CTS}$ goes low. The transmitter can only buffer one byte if it is disabled.

There is no way of knowing that the transmitter is disabled unless the $\overline{CTS}$ signal is fed into one of the I/O ports. Using the transmitter interrupt will free up the CPU to perform other functions while the transmitter is disabled or while the Transmit Buffer is full.

To enable the transmit interrupt feature Bit L5 in the Set Interrupt Register must be set. An interrupt request will not occur immediately after this bit has been set. Before any transmit interrupt request will occur a byte must be written to the Transmit Buffer. After the first byte has been written to the Transmit Buffer, a transmit interrupt request will occur, providing the transmitter is enabled.

There are three sources of transmitter interrupt requests: TBE = 1, TRE = 1, and Break-In Detect. Assuming the Break-In Detect feature is disabled, after the transmit interrupt is enabled and the first byte is written, a transmit interrupt request will be generated by TBE going active. The microprocessor can immediately write a byte to the Transmit Buffer without reading any status. However if Break-In Detect is enabled, the Status Register must be read to determine whether the transmit interrupt request was generated by Break-In Detect or TBE.

The TRE interrupt request can be used to indicate when the transmitter has completely sent all of the data. For example, using half-duplex communica-

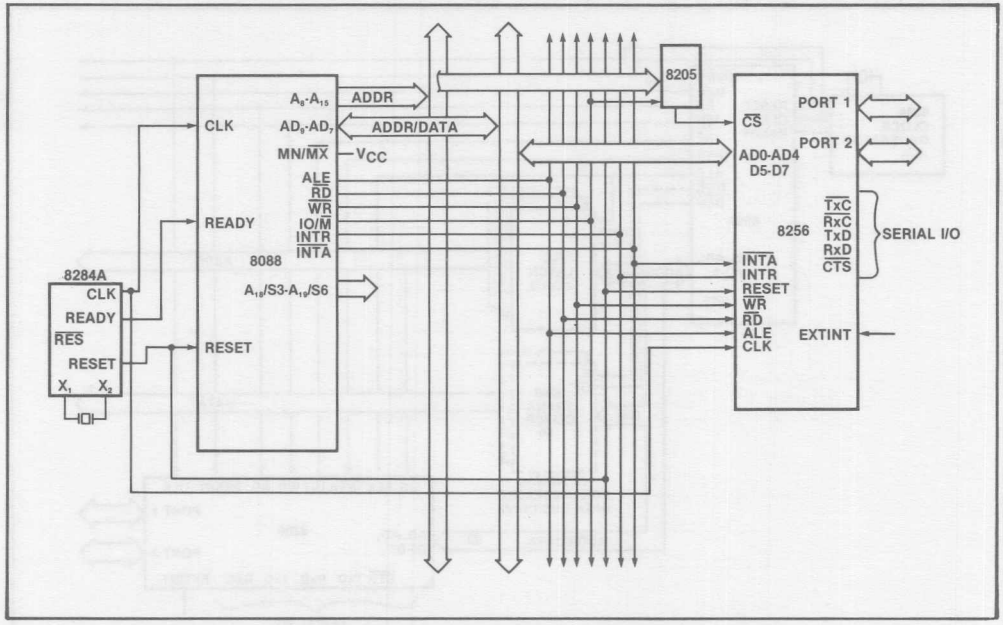


Figure 15. 8088 Min Mode/8256 Interface Multiplexed Bus

tions, all of the data written to the MUART must be transmitted before the line can be turned around. After the last byte is written, an interrupt request will be generated by TBE. If this interrupt is acknowledged without writing another byte, then the next transmitter interrupt request, TRE = 1, will indicate that the transmitter is empty and the line may be turned around.

RECEIVING

Valid data may be read from the Receive Buffer whenever the RBF bit in the Status Register is set. Reading the Receive Buffer resets the RBF status bit. The RBF bit in the Status Register can be used for polling. When the RBF bit is set, the three receive status bits, PE, OE, and FE are updated. These three status bits are reset when they are read. Therefore when the status register is read with RBF set, the three error status bit should be tested too.

If interrupts are used for serial receive data, the receiver must be enabled by setting the RxE bit in Command Register 3, and Bit L4 must be set in the Set Interrupt Register. When the receive interrupt request occurs the Receive Buffer may be read, but the status register should also be read since the receive interrupt could have been generated by the Break Detect. Also, reading the status register will indicate whether there were any errors in the received character.

Operating the Parallel Interface

Data can be transferred to or read from Port 1 and Port 2 by using the appropriate write and read operations.

LOADING PORT 1 and PORT 2

Writing to the ports transfers the data present on the data bus into the output latches. This operation is independent of the programmed I/O characteristics of the individual port pins. Writing to control or input ports has no effect on the state of the pins. Pins defined as outputs immediately assume the state which is associated with the transferred data. If inputs or control pins are reprogrammed into outputs, they assume the states stored in their output latches which were transferred by the most recent port write operation.

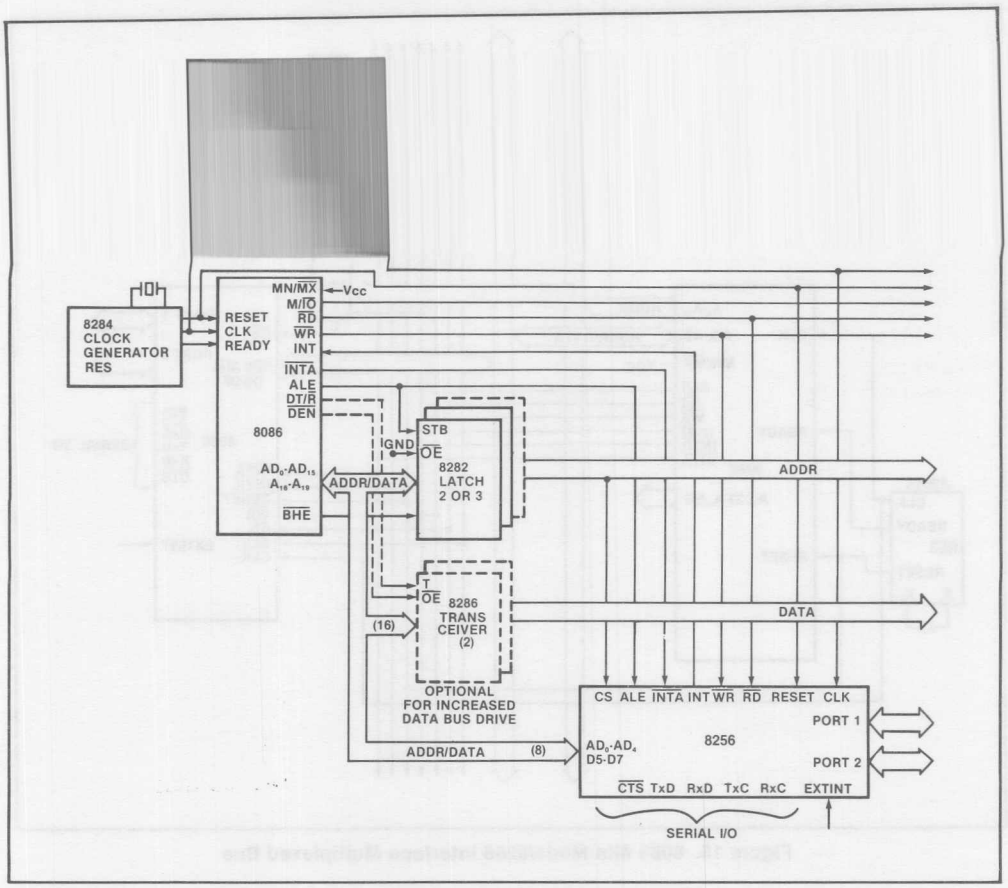


Figure 16. 8086 Min Mode/8256 Interface

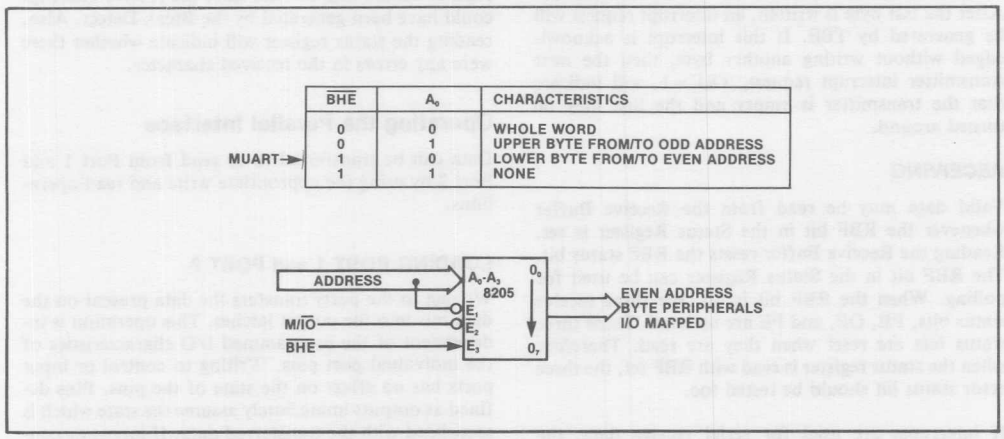


Figure 16a. Technique for Generating the MUART's Chip Select

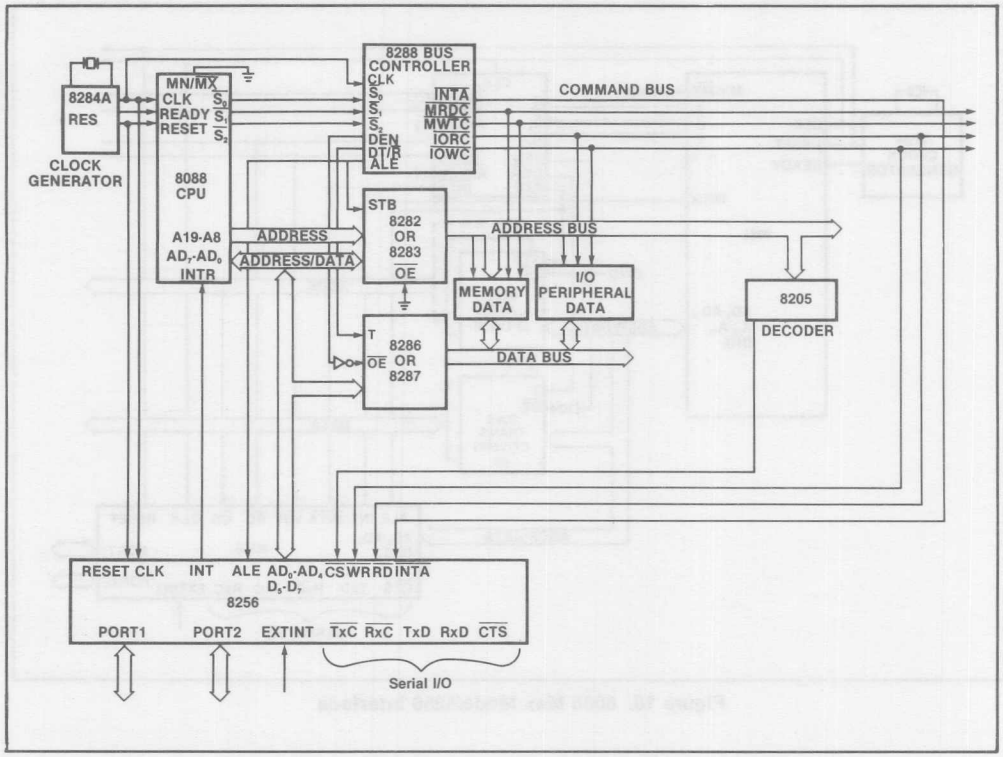


Figure 17. 8088 Max Mode/8256 Interface

READING PORT 1 AND PORT 2

Reading the ports gates the state at the pins onto the data bus if they are defined as I/O pins. A read operation transfers the contents of the associated output latches of pins P12, P13, P15, and P16, which are defined as control function pins. Reading control pins P10, P11, and P17 delivers the state of these pins.

Operating the Event Counters/Timers

The event counters/timers can be loaded with an initial value at any time. Reading event counters/timers is possible without interfering with the counting process.

LOADING EVENT COUNTERS/TIMERS

Loading event counters/timers 1-5 under their respective addresses transfers the data present on the data bus as an initial value into the addressed event counter/timer. The event counter/timer counts from the new initial value immediately following the data transfer (exception: retriggerable mode of Timer 5, or 3 and 5)

Cascaded counters/timers can be loaded with an initial value using one of two procedures:
1) Only the event counter/timer representing the most significant byte will be loaded. The event counter/timer representing the least significant byte is set to 0FFH automatically. Counting is started immediately after the data transfer.
2) The event counter/timer representing the most significant byte will be loaded, causing the least significant byte to be set to 0FFH automatically. Counting is started immediately following the data transfer. Next, the counter representing the least significant byte will be loaded and counting is started

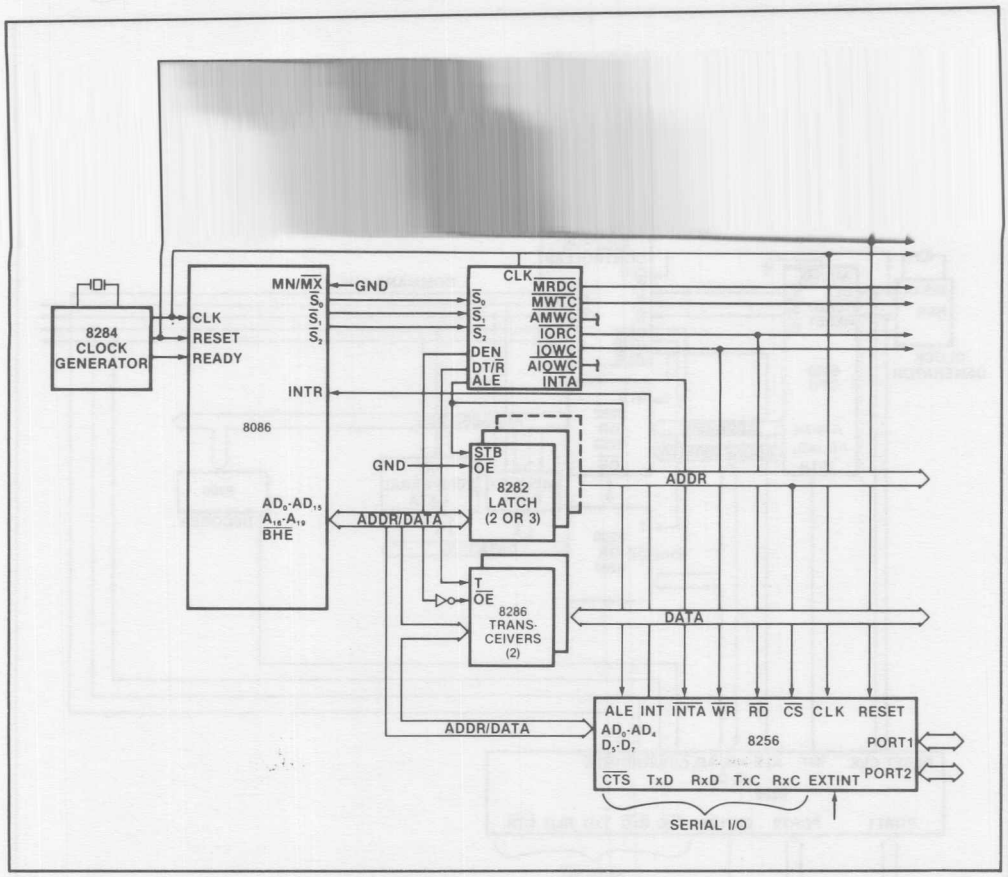


Figure 18. 8086 Max Mode/8256 Interface

again, but this time with a complete 16-bit initial value. The least significant byte of the initial value must be transferred before the counter representing the least significant byte exhibits its zero transition to prevent the most significant byte of the initial value from being decremented improperly.

In the case of an 8-bit initial value for Timer 5 or for cascaded Event Counter/Timer 3 and 5, the initial value for Timer 5 is loaded from a save register, if it is operated in retriggerable counting mode. Counting is started after an initial value has been transferred whenever a high-to-low transition occurs on Port P15.

Cascaded Event Counter/Timer 3 and 5 operating in retriggerable counting mode can be loaded directly with an initial value for Timer 5 representing the most significant byte; Event Counter/Timer 3 will be set to 0FFH automatically.

READING EVENT COUNTERS/TIMERS

Reading event counters/timers 1-5 from their respective addresses gates the counter contents onto the data bus. The counter contents gated onto the data bus remain stable during the read operation while the counter just being read continues to count. The minimum time between the two read operations from the same counter is 1 usec.

The procedure to be followed when reading cascaded event counters/timers is:

1) The event counter/timer representing the most significant byte will be read first. At this time, the least significant byte is latched into read latches.

2) When the event counter/timer representing the least significant byte is addressed, the byte stored in the read latches will be gated onto the data bus. The value stored in the read latches remains valid until it is read, the cascading condition is removed, or a write

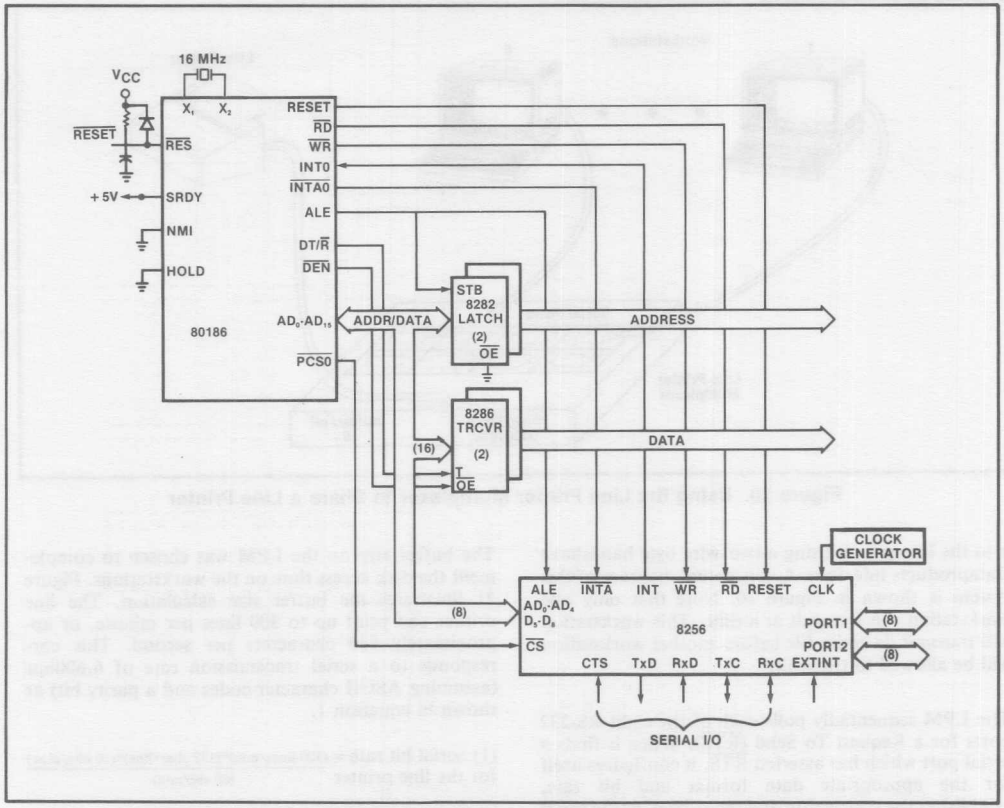


Figure 19. 80186/8256 Interface

operation affecting one of the two event counters/timers is executed.

The time between reading the most significant byte and the least significant byte must be at least 1 usec.

Note:
For cascaded event counters/timers the least significant counter/timer is latched after reading the most significant counter/timer. If the lower byte changes from 00H to 0FFH between the reading of the MSB and the latching of the LSB, the carry from the most significant event counter/timer to the least significant event counter/timer is lost.

Therefore, it is necessary to repeat the whole reading once if the value of the least significant event counter/timer is 0FFH. Doing this will avoid working with a wrong value (correct value + 255).

APPLICATION EXAMPLE

This section describes how the 8256 was designed into a Line Printer Multiplexer (LPM). This application example was chosen because it employs a majority of the MUART's features. The information in this section will be applicable to many other designs since it describes some common software and hardware aspects of using the MUART.

Description of the Line Printer Multiplexer (LPM)

The Line Printer Multiplexer allows up to eight workstations to share one printer. The workstations transmit serial asynchronous data to the LPM. The LPM receives the serial data, buffers it, then transmits

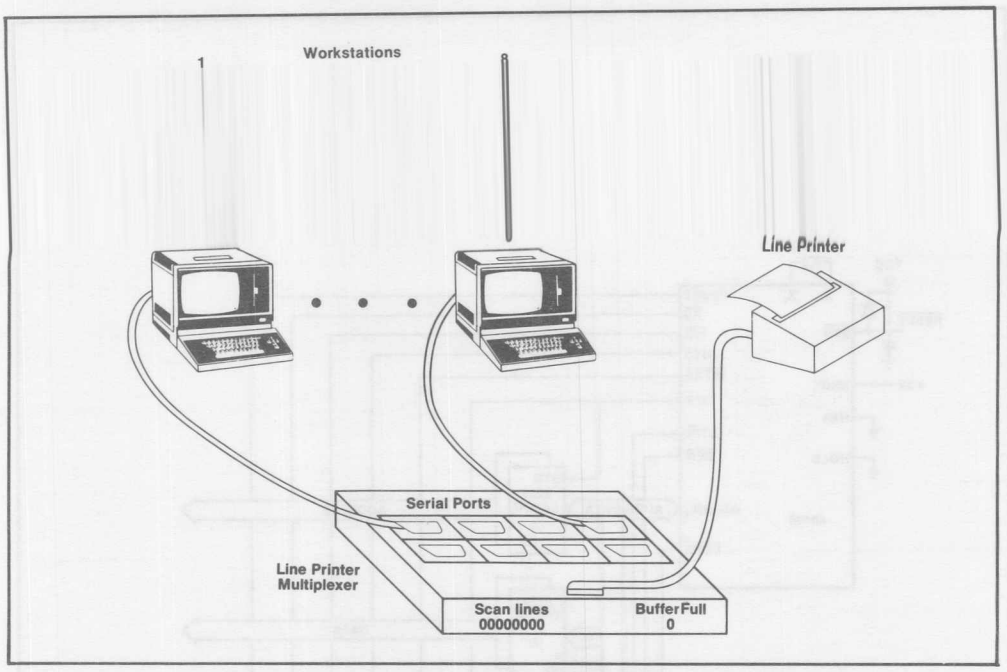


Figure 20. Using the Line Printer Multiplexer to Share a Line Printer

it to the line printer using a two-wire byte handshake Dataproducts interface. A conceptual diagram of this system is shown in Figure 20. Note that only one workstation can transmit at a time. This workstation will transmit its entire file before another workstation will be allowed to transmit.

The LPM sequentially polls each of the eight RS-232 ports for a Request To Send ($\overline{RTS}$). When it finds a serial port which has asserted $\overline{RTS}$, it configures itself for the appropriate data format and bit rate, establishes the connection and sends back to the serial port a Clear To Send ($\overline{CTS}$) which enables transmission. The LPM receives the serial asynchronous data, buffers it in a software FIFO, and transmits the data to the line printer. If the LPM detects an error in any of the serial characters it receives, it transmits an error message to the serial port and ignores the bad character. If the LPM does not receive a serial character after 18 seconds, it assumes that the transmission is complete. It transmits the final status to the serial port, and returns to scanning.

This LPM was designed to be used with single-user workstations and a 300 lines per minute line printer. These workstations are not multitasking; therefore in the middle of a file transfer when the CPU needs to reload its buffer from the disk, no serial data is transmitted. During this time the LPM is emptying its FIFO; thus, the line printer never stops printing.

The buffer size on the LPM was chosen to complement the disk access time on the workstations. Figure 21 illustrates the buffer size calculation. The line printer can print up to 300 lines per minute, or approximately 660 characters per second. This corresponds to a serial transmission rate of 6,600bps (assuming ASCII character codes and a parity bit) as shown in equation 1.

(1) Serial bit rate for the line printer $= \frac{(300 \text{ lines/min}) * (132 \text{ char/line}) * (10 \text{ bits/char})}{(60 \text{ sec/min})}$

The bottleneck in this data transfer is the line printer since the MUART and the workstations can both transmit and receive at 19.2Kbps. To realize the maximum data transfer rate of this system the LPM must guarantee that the average transfer rate to the line printer is 660 characters per second. The maximum amount of dead time that the serial port on the workstation is not transmitting, multiplied by 660 is the number of bytes which the LPM should buffer. It was determined through experimentation that it takes about 3 seconds to load 40K bytes of data from the disk into the workstation's RAM. During these 3 seconds no serial data is being sent; therefore the buffer size on the LPM should be 2K bytes. (Note: even though only a 2K byte FIFO is required, this design used an 8 Kbyte FIFO.)

To keep the LPM's buffer full the serial data rate must be greater than 6.6Kbps. The two bit rates which the

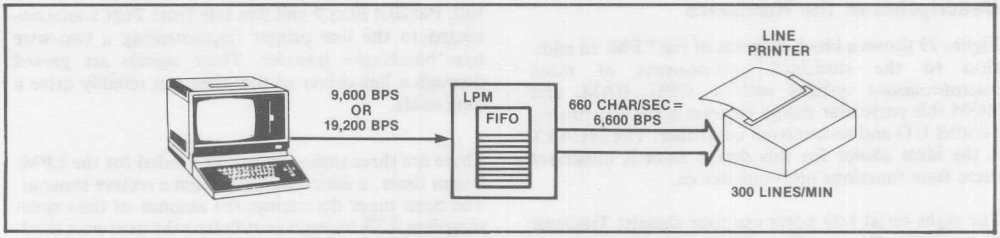


Figure 21. LPM Buffer Size Calculation

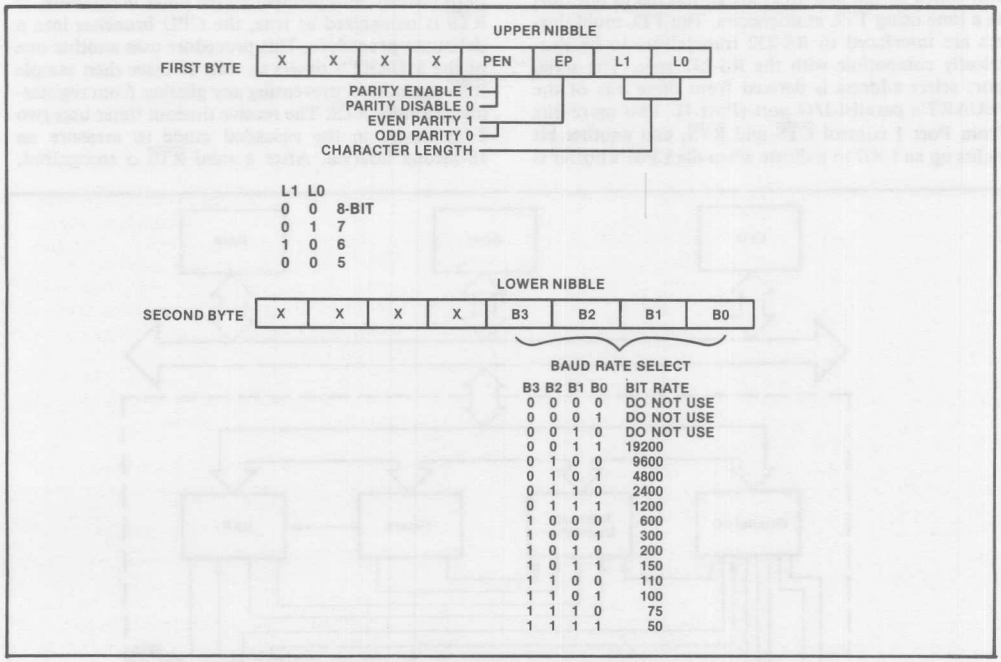


Figure 22. Programming Words Format for LPM

workstations use are 9.6Kbps and 19.2Kbps. The $\overline{CTS}$ signal is used to control the flow of the serial data so that the LPM buffer will not overflow.

Each serial port on the LPM can have a different bit rate, character length, and parity format. These parameters are programmable through the serial port. When the LPM powers up, or is reset, it expects a bit rate of 9600 bps, 7 bit characters, and odd parity. When a serial port receives an ASCII ESC character (1BH), it puts that port in the program mode. The next two bytes will program these three parameters. Only the lower nibbles of these two bytes are used, and the upper nibbles are discarded. The format of these programming words is given in Figure 22. If the word following the ESC is an ASCII NUL (0), the LPM will exit from the programming mode and not change any of its parameters.

Description of the Hardware

Figure 23 shows a block diagram of the LPM. In addition to the standard components of most microprocessor systems such as CPU, RAM, and ROM this particular design requires a UART, timers, parallel I/O and an interrupt controller. The MUART is the ideal choice for this design since it integrates these four functions onto one device.

The eight serial I/O ports use four signals: Transmit Data (TxD), Receive Data (RxD), Request To Send ($\overline{RTS}$), and Clear To Send ($\overline{CTS}$). These four signals, controlled by the MUART, are connected to one port at a time using TTL multiplexers. The TTL multiplexers are interfaced to RS-232 transceivers to be electrically compatible with the RS-232 spec. The serial port select address is derived from three bits of the MUART's parallel I/O port (Port 1). Two more bits from Port 1 control $\overline{CTS}$ and $\overline{RTS}$, and another bit lights up an LED to indicate when the LPM's buffer is full. Parallel Port 2 and two bits from Port 1 are connected to the line printer implementing a two-wire byte handshake transfer. These signals are passed through a line driver so that they can reliably drive a long cable.

There are three timing functions needed for the LPM: a scan timer, a debounce timer, and a recieve timeout. The Scan timer determines the amount of time spent sampling $\overline{RTS}$ on each port before the next port is addressed. By using one of the MUART's timers to do this function, the CPU is free to perform other functions instead of implementing the timer in software. If $\overline{RTS}$ is recognized as true, the CPU branches into a debounce procedure. This procedure uses another one of the MUART's timers to wait 10 msec then sample $\overline{RTS}$ again, thus preventing any glitches from registering as a false $\overline{RTS}$. The receive timeout timer uses two 8-bit timers in the cascaded mode to measure an 18-second interval. After a valid $\overline{RTS}$ is recognized,

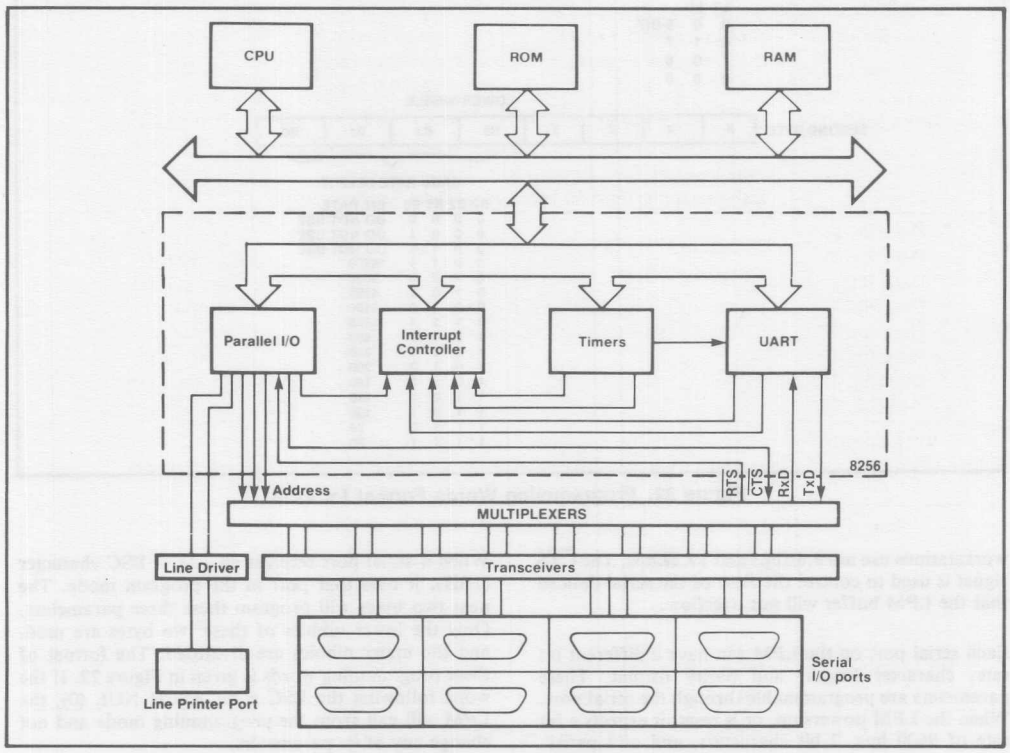


Figure 23. Functional Block Diagram of the Line Printer Multiplexer

the LPM sends back a $\overline{CTS}$ and initializes the receive timeout timer for 18 seconds. Each time a character is received by the LPM, this timer is reinitialized. If this timer times out, the LPM considers the transmission complete and returns to scanning.

The schematic diagram of the LPM is shown in Figure 24. The CPU is an 8088 used in the min mode. It is interfaced directly to the 8256. An 8282 latch is employed in the system so that nonmultiplexed bus memory can be used. A 2716 holds the entire program, and six 2016s (2K x 8 static RAMs) are used to store the buffer, temporary data, stack area, and interrupt vector table. The 2716 is located in the upper 2K of the 8088 address space (FF800-FFFFFH) so that the reset vectors can be stored starting at location FFFF0H. The RAM address space spans 0-2FFFH so that the interrupt vector table can be stored starting at location 0. The MUART is I/O mapped and its registers occupy even addresses from 0 to 1EH. Using an 8088 CPU the MUART must be placed in the 8086 mode since the $\overline{INTA}$ signal is used; hence the register addresses are all even numbers.

The line printer used provides a choice of two standard parallel interfaces: Centronics or Dataproducts. The Centronics interface uses a two-wire handshake pulsed strobe where the transmitter asserts a complete strobe pulse before an acknowledge is received. The Dataproducts interface is an interlocking two-wire handshake. The Dataproducts interface was chosen since it is directly compatible with the MUART's two-wire byte handshake. The MUART could also be connected to the Centronics interface; however, additional hardware would be necessary to generate the pulsed strobe for correct interrupt operation. Figure 25 shows the timing of the Dataproducts interface and Table 6 lists the connector pin configuration.

Table 6. Dataproducts Interface Line Functions

Signal	Description	Connector Pin
Data Request	Sent by printer to synchronize data transmission. When true, requests a character. Remains true until **Data strobe** is received, then goes false within 100 nsec.	E(return C)
Data Strobe	Sent by user system to cause printer to accept information on data lines. Should remain true until printer drops **Data Request** line. Data lines must stabilize for at least 50 nsec before **Data Strobe** is sent.	j(return m)
Data Bit 1	Bit 8 controls optional character set Refer to *Commands and Formats.*	B(return D)
Data Bit 2		F(return J)
Data Bit 3		L(return N)
Data Bit 4		R(return T)
Data Bit 5		V(return X)
Data Bit 6		Z(return b)
Data Bit 7		n(return k)
Data Bit 8		h(return e)
VFU Control (PI)	Optional control from user system. Used for VFU control. Data Request/Strobe timing is same as for data lines.	p(return s)
Ready	Sent to user system by printer. True when no Check condition exists.	CC(return EE)
On Line	Sent to user system by printer. True when **Ready** line is true and operator has activated ON LINE Pushbutton. Enables interface activity.	y(return AA)
Interface Verify	Jumper in printer connector. Continuity informs user system that connector is properly seated.	x to v
+5V	Supply voltage for Exerciser only.	HH

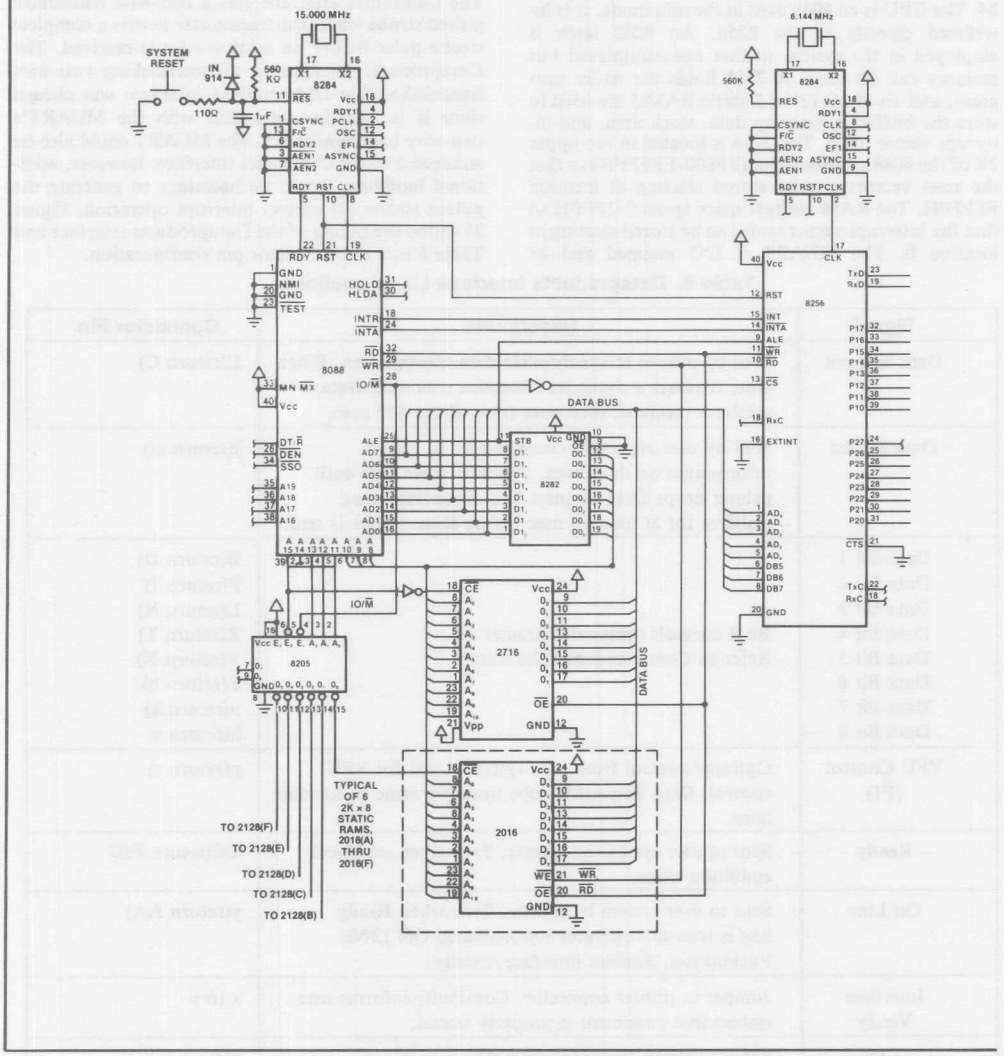


Figure 24. Schematic of LPM

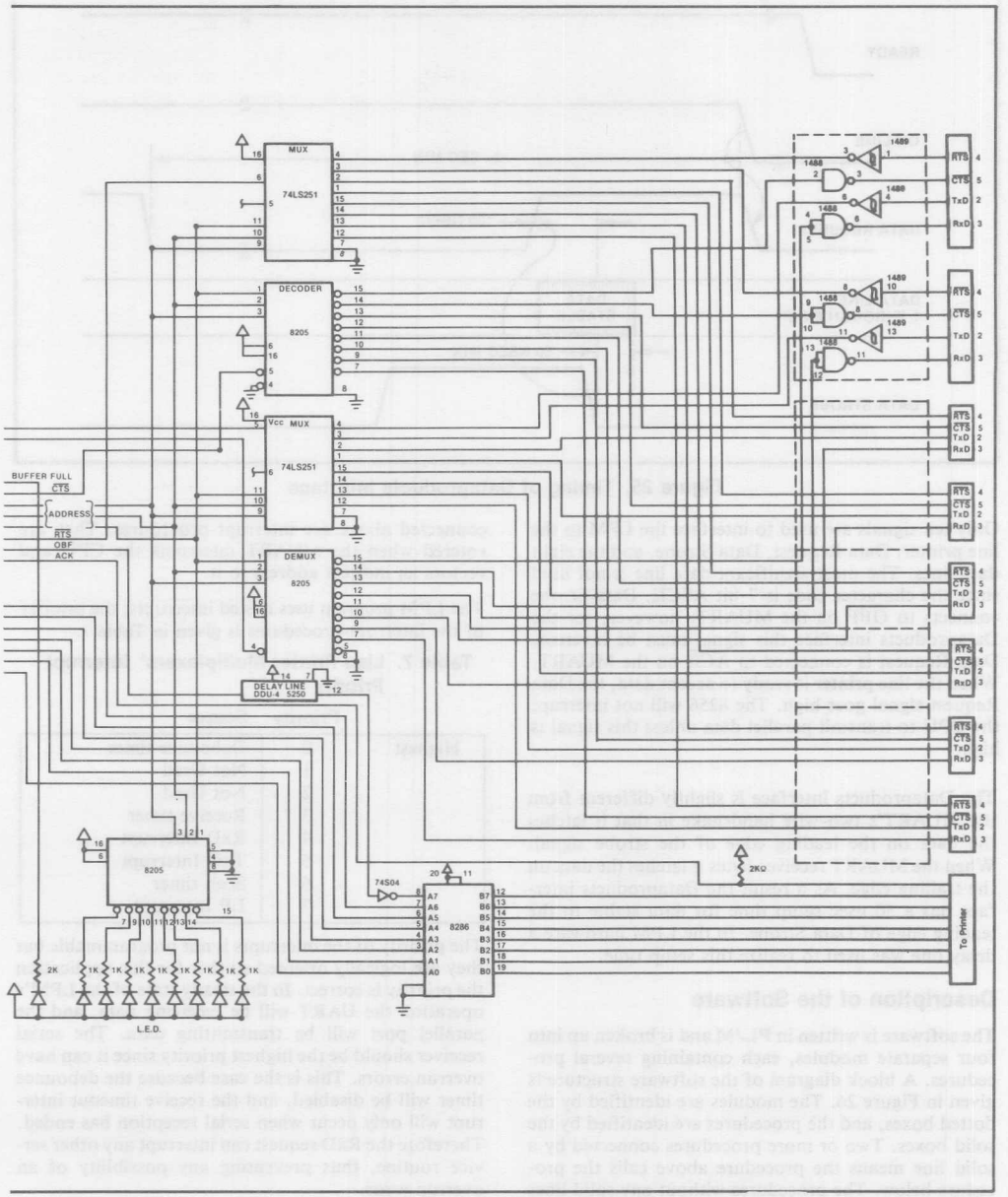


Figure 24. Schematic of LPM (Continued)

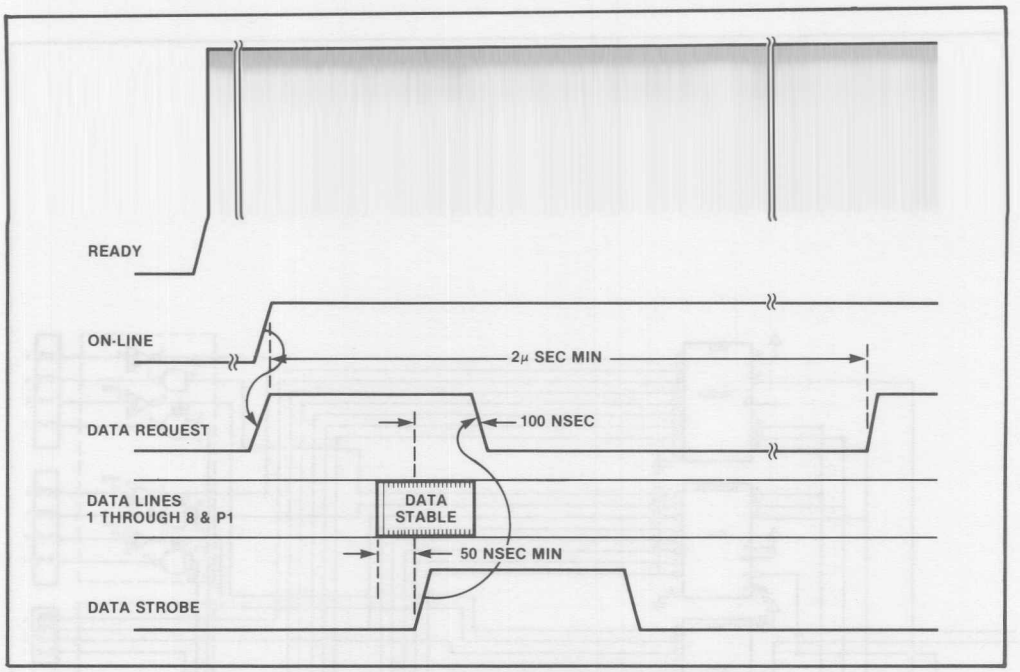


Figure 25. Timing of Dataproducts Interface

Only ten signals are used to interface the LPM to the line printer: Data Request, Data Strobe, and the eight data lines. The most significant data line is not used since the character code is 7-bit ASCII. Data Strobe connects to $\overline{OBF}$ on the MUART; however, for the Dataproducts interface this signal must be inverted. Data Request is connected to $\overline{ACK}$ on the MUART. When the line printer is ready to accept data, the Data Request signal goes high. The 8256 will not interrupt the CPU to transmit parallel data unless this signal is high.

The Dataproducts interface is slightly different from the MUART's two-wire handshake in that it latches the data on the leading edge of the strobe signal. When the MUART receives bytes it latches the data on the trailing edge. As a result the Dataproducts interface has a 50 nsec setup time for data stable to the leading edge of Data Strobe. In the LPM hardware a delay line was used to realize this setup time.

Description of the Software

The software is written in PL/M and is broken up into four separate modules, each containing several procedures. A block diagram of the software structure is given in Figure 26. The modules are identified by the dotted boxes, and the procedures are identified by the solid boxes. Two or more procedures connected by a solid line means the procedure above calls the procedure below. The procedures without any solid lines connected above are interrupt procedures. They are entered when the MUART interrupts the CPU and vectors an indirect address to it.

The LPM program uses nested interrupts; the priority of the interrupt procedures is given in Table 7.

Table 7. Line Printer Multiplexers' Interrupt Priority

	Priority	Source
Highest	0	Debounce timer
	1	Not Used
	2	Not Used
	3	Receive timer
	4	RxD Interrupt
	5	TxD Interrupt
	6	Scan timer
	7	LP Interrupt

The priority of the interrupts is not programmable but they are logically oriented so that for this application the priority is correct. In the steady state of the LPM's operation the UART will be receiving data, and the parallel port will be transmitting data. The serial receiver should be the highest priority since it can have overrun errors. This is the case because the debounce timer will be disabled, and the receive timeout interrupt will only occur when serial reception has ended. Therefore the RxD request can interrupt any other service routine, thus preventing any possibility of an overrun error.

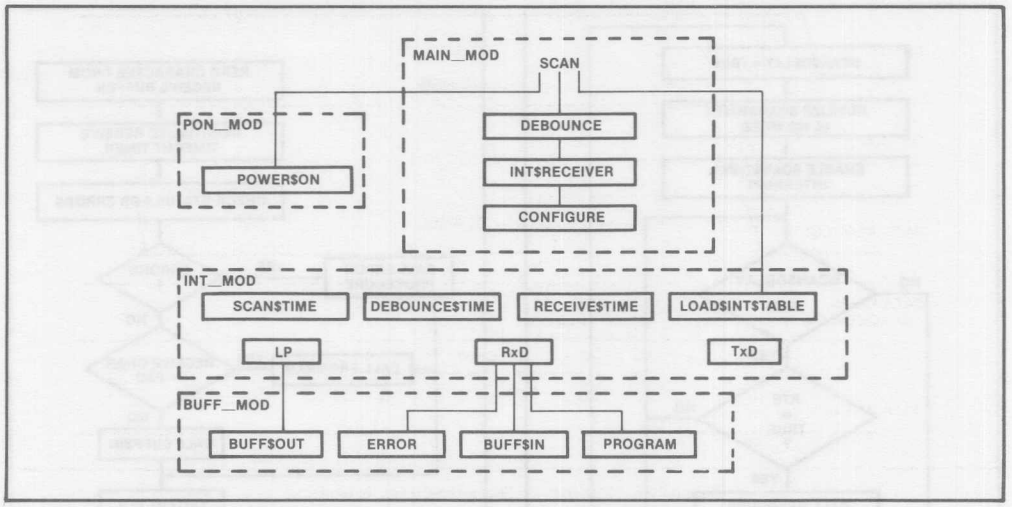


Figure 26. Block Diagram of LPM Software Structure

On power-up the CPU branches from 0FFFF0H to the INITCODE routine which is included in the machine code by the MDS locater utility. INITCODE initializes the 8088's segment registers, stack pointer, and instruction pointer, then it disabled interrupts and jumps into MAIN_MOD. The first executable instruction in MAIN_MOD calls POWER$ON, which initializes the MUART, flags, variables, and arrays. The MAIN_MOD calls LOADINTTABLE, which initializes the interrupt vector table. The CPU's interrupt is then enabled and the program enters into a DO FOREVER loop which scans the eight serial ports for an $\overline{\text{RTS}}$.

There are three software functions which employ the MUART's timers and interrupt controller to measure time intervals: SCAN, debounce, and INIT$RECEIVER. DEBOUNCE and INIT$RECEIVER procedures, employ the MUART's timers and interrupt controller to measure time intervals. The CPU remains in a loop for a specific amount of time before it proceeds with the next section of code. In this loop the CPU is waiting for a global status flag to change while servicing any interrupts which may occur. When the appropriate timer interrupt occurs, the interrupt service routine will set the global flag which causes the CPU to exit the loop and proceed to the next section of code. An example can be seen from the scan flow chart in Figure 27.

The first thing the program does before entering the loop is set the flag (in this case SCAN$DELAY) TRUE. The timer is initialized and the loop is entered. As long as SCAN$DELAY is TRUE the CPU will continue to sample $\overline{\text{RTS}}$. If $\overline{\text{RTS}}$ remains false for more than 100 msec, the timer interrupts the CPU and the interrupt service routine sets SCAN$DELAY FALSE. This causes the CPU to exit the loop and address the next port. The process is then repeated. If $\overline{\text{RTS}}$ becomes true while it is being sampled, the DEBOUNCE procedure is called.

DEBOUNCE does nothing more than wait 10 msec and sample $\overline{\text{RTS}}$ again using the same technique discussed above. If $\overline{\text{RTS}}$ is still valid INIT$RECEIVER is called, otherwise the CPU returns to scan.

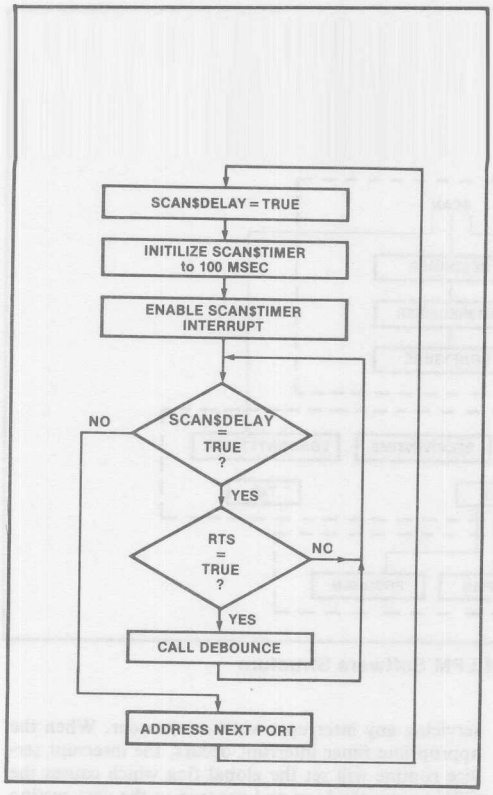


Figure 27. Scan Flow Chart

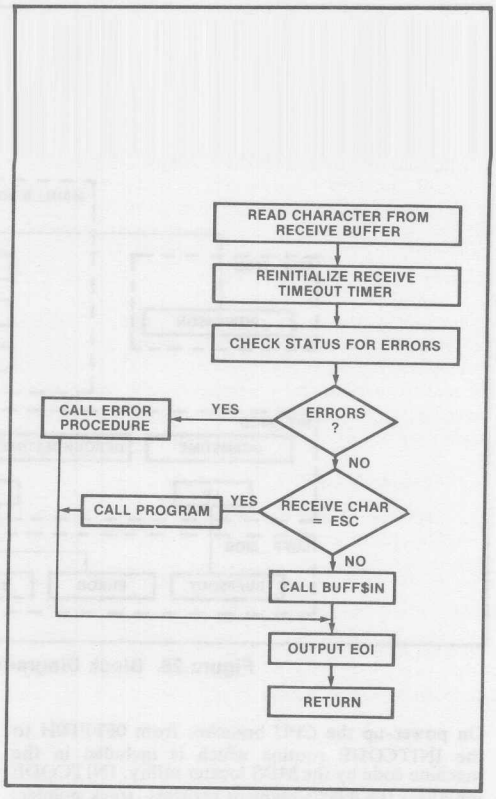


Figure 28. RxD Interrupt Procedure Flow Chart

INIT$RECEIVER calls CONFIGURE which programs the MUART for the bit rate, number of bits in a character, and parity format. This information is stored in an array called SERIAL$FORMAT, which contains a byte for each port. The bytes in the SERIAL$FORMAT array have the same bit definition as the two nibbles in the programming words in Figure 22. Upon returning to INIT$RECEIVER the receiver is enabled, the receive timeout timer is initialized, and the timer and receiver interrupts are enabled. CTS on the serial port is then set true, and the CPU enters a loop which does nothing except wait for 18 seconds. If no characters are received within 18 seconds, the receive timeout interrupt occurs and the loop flag is set false, which causes the CPU to exit the loop. If a character is received, a receive interrupt occurs, and the CPU vectors into the RxD interrupt service routine.

Figure 28 shows a flow chart of the RxD interrupt service routine. This routine begins by reading the receive buffer and reinitializing the receive timeout timer. There are two conditions to check for before the character can be inserted into the FIFO. First, if there are any errors in the received character, an ERROR procedure is called which reports back to the serial port what the error condition was. The character in error is discarded and the routine returns. The other condition is that if the received character is an ASCII ESC, the PROGRAM procedure is called. If neither one of these conditions occurs, the character is placed in the FIFO by the BUFF$IN procedure.

The LP interrupt routine is entered when the byte handshake interrupt request is acknowledged. This routine simply calls the BUFF$OUT procedure, which extracts a byte out of the FIFO. BUFF$OUT returns the byte to the LP interrupt procedure, which then writes it to Port 2. One small problem with getting the handshake interrupt going is that the first byte has to be written to Port 2 before the first handshake interrupt will occur. The problem is that the line printer may not be ready for the first byte. This would be indicated by DATA REQUEST being low. If the byte was written to the LP while DATA REQUEST is low, it would be lost. Note that if the handshake interrupt is enabled while DATA REQUEST is low, then DATA REQUEST goes high, the interrupt will occur without

writing the first byte. There are several ways to solve this problem. Port 1 can be read to find out what the state of the DATA REQUEST line is. If DATA REQUEST is low, the CPU can simply wait for the interrupt without writing the first byte. If DATA REQUEST is high, then the first data byte may be written. Another solution would be to write a NUL character as the first byte to Port 2. If DATA REQUEST is low, then a worthless character is lost. If DATA REQUEST is high, the NUL character would be sent to the line printer; however, it is not printed since NUL is a nonprintable character. The LPM program uses the NUL character solution.

BUFFER MANAGEMENT

The FIFO implementation uses an 8K byte array to store the characters. There are two pointers used as indexes in the array to address the characters: IN$POINTER and OUT$POINTER. IN$POINTER points to the location in the array which will store the next byte of data inserted. OUT$POINTER points to the next byte of data which will be removed from the array. Both IN$POINTER and OUT$POINTER are declared as words. Figure 29 illustrates the FIFO in a block diagram.

The BUFF$IN procedure receives a byte from the RxD interrupt routine and stores it in the array location pointed to by IN$POINTER, then IN$POINTER is incremented. Similarly, when BUFF$OUT is called

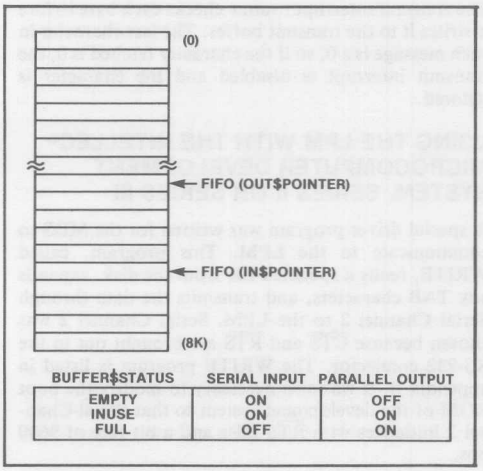


Figure 29. FIFO Structure and Status

by the LP interrupt routine, the byte in the array pointed to by OUT$POINTER is read. OUT$POINTER is incremented, and the byte which was read is passed back to the LP interrupt routine. Since IN$POINTER and OUT$POINTER are always incremented, they must be able to roll over when they hit the top of the 8K byte address space. This is done by clearing the upper three bits of each pointer after it is incremented.

IN$POINTER and OUT$PONTER not only point to the locations in the FIFO, they also indicate how many bytes are in the FIFO and whether the FIFO is full or empty. When a character is placed into the FIFO and IN$POINTER is incremented, the FIFO is full if IN$POINTER equals OUT$POINTER. When a character is read from the FIFO and OUT$-POINTER is incremented, the FIFO is empty if OUT$POINTER equals IN$POINTER. If the buffer is neither full nor empty, then it is in use. A byte called BUFFER$STATUS is used to indicate one of these three conditions.

The software uses the buffer status information to control the flow into and out of the FIFO. When the FIFO is empty the handshake interrupt must be turned off. When the FIFO is full, $\overline{CTS}$ must be sent false so that no more data will be received. If the buffer status is in use, $\overline{CTS}$ is true and the handshake interrupt is enabled.

Figure 30 shows the flow chart of the BUFF$IN procedure. The BUFF$IN procedure begins by checking the BUFFER$STATUS. If it is empty and the character to be inserted into the FIFO is a CR or LF, the handshake interrupt is enabled, a NUL character is output, and the BUFFER$STATUS is set to IN-USE. The character passed to BUFF$IN from RxD is put into the FIFO. If the FIFO is now full, the BUF-FER$STATUS is set to FULL, $\overline{CTS}$ is set false, and the buffer full LED is turned on.

Figure 31 shows the flow chart of the BUFF$OUT procedure. After the character is read from the FIFO, the FIFO is tested to determine if it is empty. If it is not empty, the BUFFER$STATUS is FULL and there are 200 bytes available in the FIFO, serial data reception is reenabled, and the FIFO fills again. While data is being received from the workstation, $\overline{CTS}$ toggles high and low, filling up and emptying the last 200 bytes in the FIFO. Referring to the top of the flow chart (FIFO empty test) if it's empty, the BUF-FER$STATUS is set to EMPTY, and the handshake interrupt is disabled. During this time all interrupts

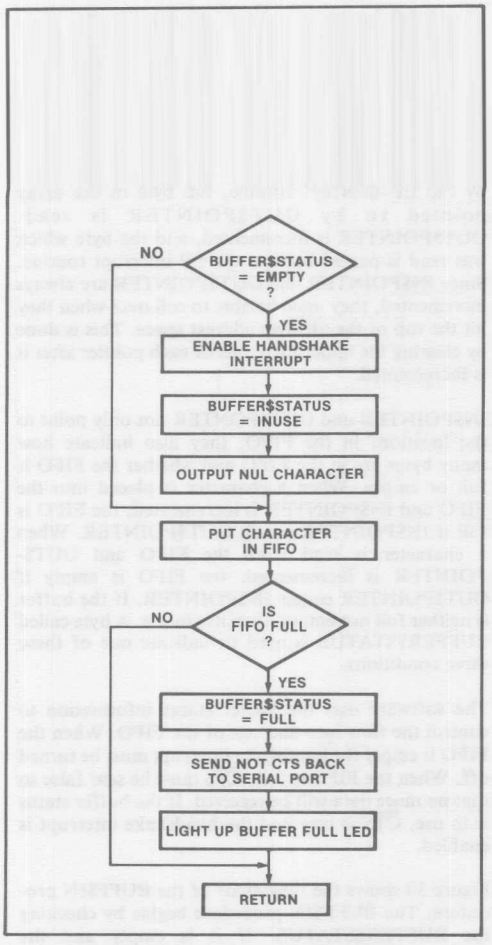


Figure 30. Flow Chart of the BUFF$IN Procedure

are disabled at the CPU. (Remember that the RxD interrupt routine can interrupt the LP and BUFF$OUT procedures since it has a higher priority, and the MUART is in the nested mode.)

If the CPU interrupt was not disabled during this time, the following events could occur which would cause the LPM to crash. Assume that the RxD interrupt occured where the asterisk is in the flow chart, after BUFFER$STATUS is set to EMPTY. The BUFF$IN procedure would set BUFFER$STATUS to INUSE and enable the handshake interrupt. When the RxD interrupt routine returned to BUFF$OUT, the handshake interrupt is disabled, but the BUFFER$STATUS is INUSE. The handshake interrupt could never be reenabled, and the FIFO would fill up. This is known as a critical section of code. Suspicion should arise for a critical section of code when two or more nested interrupt routines can affect the same status. One solution is to disable the interrupt flag at the CPU while the status and conditional operations are being modified.

The flow chart for the TxD interrupt procedure is given in Figure 32. For this program five different messages can be transmitted, and they are stored in ROM. It is possible to download the messages into a dedicated RAM buffer; however, the RAM buffer would have to be as large as the largest message. A more efficient way to transmit the messages is to read them from ROM. In this case the address of the first byte of the message would have to be accessible by the transmit interrupt procedure. Since parameters cannot be passed to interrupt procedures, this message pointer is declared PUBLIC in one module and EXTERNAL in the other modules.

To get the transmit interrupt started, the first byte of the message must be written to the transmit buffer. When a section of code decides to transmit a message serially, it loads the global message pointer with the address of the first byte of the message, enables the transmit interrupt, and calls the TxD interrupt procedure. Calling the TxD interrupt procedure writes the first byte to the transmit buffer to initiate transmit interrupts. This can be done by calling PL/M's built-in procedure CAUSE$INTERRUPT.

The transmit interrupt routine checks each byte before it writes it to the transmit buffer. The last character in each message is a 0, so if the character fetched is 0, the transmit interrupt is disabled and the character is ignored.

USING THE LPM WITH THE INTELLEC® MICROCOMPUTER DEVELOPMENT SYSTEM, SERIES II OR SERIES III

A special driver program was written for the MDS to communicate to the LPM. This program, called WRITE, reads a specified file from the disk, expands any TAB characters, and transmits the data through Serial Channel 2 to the LPM. Serial Channel 2 was chosen because $\overline{CTS}$ and $\overline{RTS}$ are brought out to the RS-232 connector. The WRITE program is listed in appendix B. It was also necessary to modify the boot ROM of the development system so that Serial Channel 2 initializes with $\overline{RTS}$ false and a bit rate of 9600 bps.

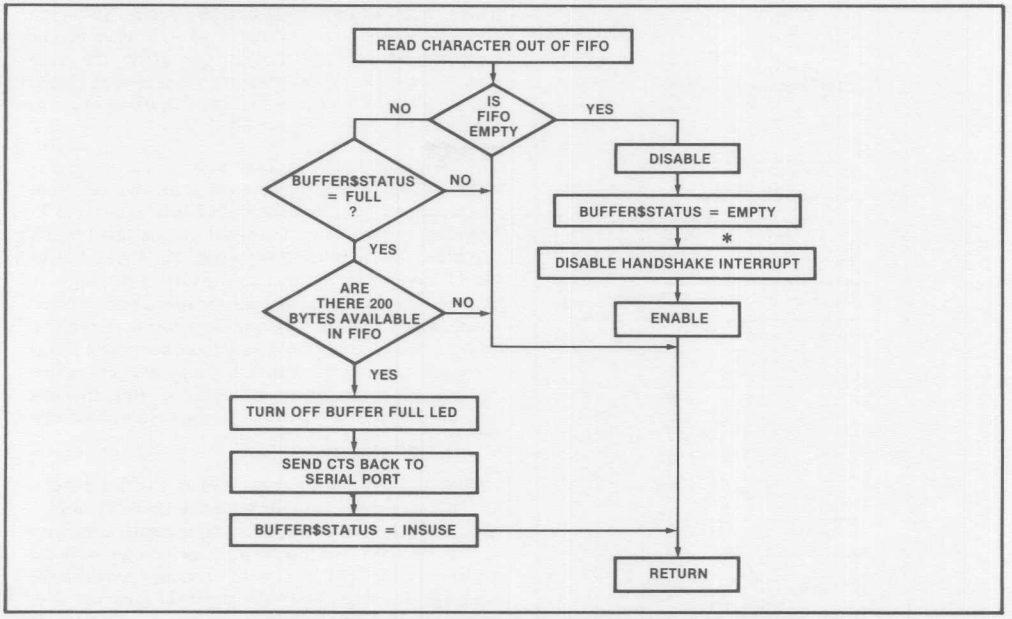


Figure 31. Flow Chart of the BUFF$OUT Procedure

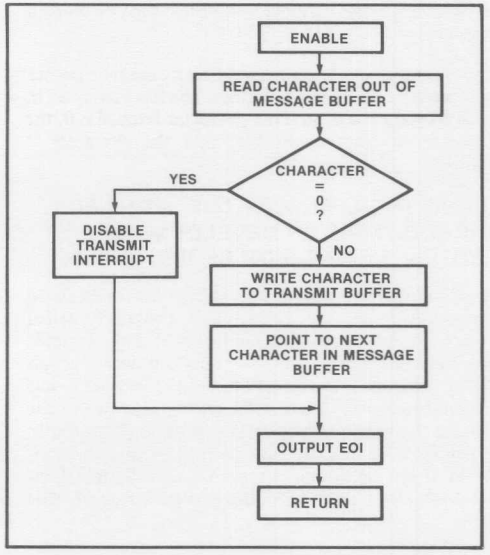
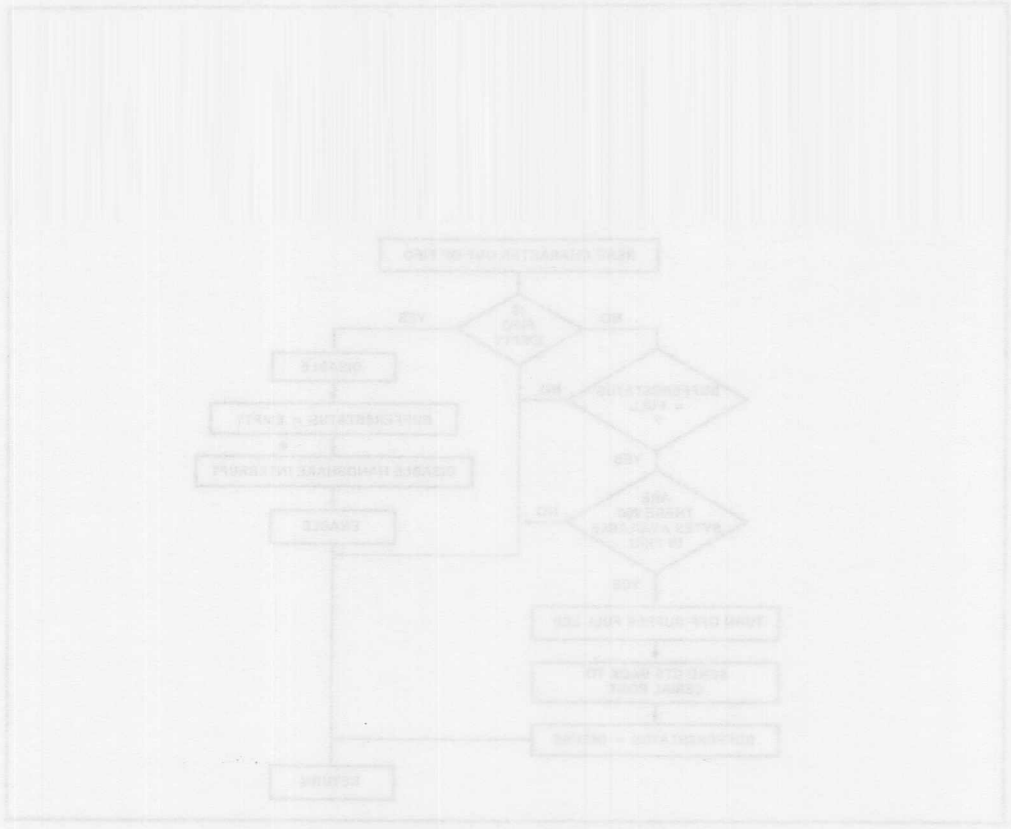
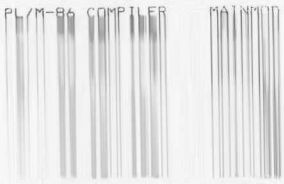


Figure 32. Flow Chart for TxD Interrupt Procedure



APPENDIX A
LISTING OF THE LINE PRINTER MULTIPLEXER SOFTWARE




```
SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE MAINMOD
OBJECT MODULE PLACED IN :F1:MAIN.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:MAIN.SRC


                /*************************************************************************
                 *                                                                       *
                 *            MAIN MODULE FOR THE LINE PRINTER MULTIPLEXER               *
                 *                                                                       *
                 *************************************************************************/

                $DEBUG
  1             MAIN$MOD: DO;

                /*************************************************************************
                * PORT 1 BIT CONFIGURATION                                               *
                *                                                                        *
                *  BUFFER FULL   CTS      ADDRESS    RTS   TWO WIRE HANDSHAKE           *
                *       B7        B6     B5 B4 B3    B2       B1 B0                     *
                *************************************************************************/

  2   1         DECLARE LIT             LITERALLY       'LITERALLY',
                        TRUE            LIT             'OFFH',
                        FALSE           LIT             '0',
                        FOREVER         LIT             'WHILE 1',

                        CMD$1           LIT             '0',    /*8256 REGISTERS*/
                        CMD$2           LIT             '2',
                        CMD$3           LIT             '4',
                        MODE            LIT             '6',
                        PORT$1$CTRL     LIT             '8',
                        SET$INT         LIT             '0AH',
                        INT$EN          LIT             '0AH',
                        RST$INT         LIT             '0CH',
                        INT$ADDR        LIT             '0CH',
                        TX$BUFF         LIT             '0EH',
                        RX$BUFF         LIT             '0EH',
                        PORT$1          LIT             '10H',
                        PORT$2          LIT             '12H',
                        DEBOUNCE$TIMER  LIT             '14H',
                        SCAN$TIMER      LIT             '1AH',
                        RECEIVE$TIMER   LIT             '1CH',
                        STATUS$REG      LIT             '1EH',

                        SCAN$INT        LIT             '40H',
                        DEBOUNCE$INT    LIT             '01H',
                        RECEIVER$INT    LIT             '10H',
                        TIME$OUT$INT    LIT             '08H',
                        TRANSMIT$INT    LIT             '20H',

                        EMPTY           LIT             '0',
                        INUSE           LIT             '1',
                        FULL            LIT             '2',

                        RTS             LIT             '(INPUT(PORT$1) AND 04H)',
```

```
PL/M-86 COMPILER    MAINMOD

                    BEGIN           LABEL           PUBLIC,

                    TEMP            BYTE,
                    SCAN$DELAY      BYTE            PUBLIC,
                    DEBOUNCE$DELAY  BYTE            PUBLIC,
                    RECEIVE$DELAY   BYTE            PUBLIC,
                    PORT$PTR        BYTE            PUBLIC,
                    SERIAL$FORMAT(8)BYTE            PUBLIC, /* PEN EP L1 L0 B3 B2 B1 B0 */

                    MESSAGE$PTR     POINTER         EXTERNAL,
                    J               BYTE            EXTERNAL,
                    OK(1)           BYTE            EXTERNAL,
                    BUFFER$STATUS   BYTE            EXTERNAL;

                /*****************************************************************
                 *                  EXTERNAL PROCEDURE DECLARATIONS              *
                 *****************************************************************/

  3   1         POWER$ON: PROCEDURE EXTERNAL;
  4   2         END POWER$ON;

  5   1         LOAD$INT$TABLE: PROCEDURE EXTERNAL;
  6   2         END LOAD$INT$TABLE;

                /*****************************************************************
                 *      SET THE BIT RATE AND DATA FORMAT FOR THE SERIAL PORT     *
                 *****************************************************************/

  7   1         CONFIGURE: PROCEDURE ; /*Initialize bit rate and data format*/
  8   2         TEMP=SERIAL$FORMAT(SHR(PORT$PTR,3));
  9   2         OUTPUT(CMD$1)=((SHL(TEMP,2) AND 0C0H) OR 03H);
 10   2         OUTPUT(CMD$2)=(TEMP OR 30H);
 11   2         END CONFIGURE;

                /*****************************************************************
                 *                  INITIALIZE SERIAL RECEIVER                   *
                 *****************************************************************/

 12   1         INIT$RECEIVER:  PROCEDURE;
 13   2         CALL CONFIGURE;                 /*Initialize 8256 serial port*/
 14   2         RECEIVE$DELAY=TRUE;
 15   2         OUTPUT(CMD$3)=0C0H;             /*Enable serial receiver*/
 16   2         OUTPUT(RECEIVE$TIMER)=70;       /*18 second TIME$OUT*/
 17   2         OUTPUT(SET$INT)=18H;            /*Enable RECEIVER and TIME$OUT interrupts*/
 18   2         IF (BUFFER$STATUS<>FULL)
                        THEN
 19   2           OUTPUT(PORT$1)=(INPUT(PORT$1) AND 0BFH); /*Send CTS TRUE*/

 20   2             DO WHILE RECEIVE$DELAY=TRUE; /* Wait here while receiving serial data */
 21   3             END;

                        /* After 18 seconds of not receiving a character, proceed */

 22   2         OUTPUT(SET$INT)=TRANSMIT$INT;     /* Send the terminating message */
 23   2         J=0;
 24   2         MESSAGE$PTR= @OK(0);
 25   2         CAUSE$INTERRUPT (45H);
```

```
PL/M-86 COMPILER    MAINMOD


  26   2      OUTPUT(PORT$1)=(INPUT(PORT$1) OR 40H); /*Send CTS FALSE*/
  27   2      OUTPUT(RST$INT)=18H;              /*Clear RECEIVER and TIMER Interrupts*/
  28   2      OUTPUT(CMD$3)=40H;                /*Disable serial receiver*/
  29   2      END INIT$RECEIVER;

              /*****************************************************************
               *                          DEBOUNCE RTS                         *
               *****************************************************************/

  30   1      DEBOUNCE: PROCEDURE;
  31   2      DEBOUNCE$DELAY=TRUE;
  32   2      OUTPUT(DEBOUNCE$TIMER)=10; /* 10 msec debounce time delay */
  33   2      OUTPUT(SET$INT)=DEBOUNCE$INT;
  34   2          DO WHILE DEBOUNCE$DELAY=TRUE;
  35   3          END;
  36   2      IF RTS=0 THEN CALL INIT$RECEIVER;
  38   2      END DEBOUNCE;

              /*****************************************************************
               *                        BEGIN MAIN PROGRAM                     *
               *****************************************************************/


  39   1      BEGIN: CALL POWER$ON;

  40   1      CALL LOAD$INT$TABLE;

  41   1      ENABLE;

  42   1      DO FOREVER;

  43   2      SCAN$DELAY=TRUE;
  44   2      OUTPUT(SCAN$TIMER)=100;     /*Spend 100 msec on each serial port sampling RTS*/
  45   2      OUTPUT(SET$INT)=SCAN$INT;

  46   2          DO WHILE SCAN$DELAY=TRUE;    /*Sample RTS*/
  47   3            IF RTS=0
                       THEN
  48   3              CALL DEBOUNCE;
  49   3          END;


  50   2      TEMP=INPUT(PORT$1);               /*Increment PORT$PTR*/
  51   2      PORT$PTR=TEMP AND 38H;
  52   2      TEMP=TEMP AND (NOT 38H);
  53   2      PORT$PTR=(PORT$PTR+8) AND 38H;

  54   2      OUTPUT(PORT$1)=TEMP OR PORT$PTR; /*Look at next serial port*/
  55   2      END; /*DO FOREVER*/
  56   1      END MAIN$MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 011CH    284D


PL/M-86 COMPILER    MAINMOD


     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 000DH     13D
     MAXIMUM STACK SIZE = 000CH     12D
     159 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
PL/M-86 COMPILER     INTMOD

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE INTMOD
OBJECT MODULE PLACED IN :F1:INT.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:INT.SRC


                /*****************************************************************
                 *                                                               *
                 *        INTERRUPT MODULE:  CONTAINS ALL INTERRUPT ROUTINES     *
                 *                           PLUS LOAD INTERRUPT TABLE PROCEDURE *
                 *                                                               *
                 *****************************************************************/

                $DEBUG
   1            INT$MOD: DO;
                $NOLIST

   3    1       DECLARE
                        ESC             LIT             '1BH',
                        SCAN$DELAY      BYTE            EXTERNAL,
                        DEBOUNCE$DELAY  BYTE            EXTERNAL,
                        RECEIVE$DELAY   BYTE            EXTERNAL,
                        MESSAGE$PTR     POINTER         EXTERNAL,
                        J               BYTE            EXTERNAL,

                /*****************************************************************
                 *                 MESSAGES SENT TO SERIAL PORTS                 *
                 *****************************************************************/

                        OK (*) BYTE PUBLIC DATA ('TRANSMISSION COMPLETE', 0AH, 0DH, 00),
                        BREAK (*) BYTE PUBLIC DATA ('BREAK DETECT ERROR', 0AH, 0DH, 00),
                        PARITY (*)BYTE PUBLIC DATA ('PARITY ERROR DETECTED', 0AH, 0DH, 00),
                        FRAME (*) BYTE PUBLIC DATA ('FRAMING ERROR DETECTED', 0AH, 0DH, 00),
                        OVER$RUN(*)BYTE PUBLIC DATA('OVER RUN ERROR DETECTED', 0AH, 0DH, 00);

                /*****************************************************************
                 *       EXTERNAL PROCEDURES CALLED BY THE INTERRUPT ROUTINES     *
                 *****************************************************************/
   4    1       ERROR: PROCEDURE (STATUS) EXTERNAL;
   5    2       DECLARE STATUS  BYTE;
   6    2       END ERROR;

   7    1       PROGRAM: PROCEDURE EXTERNAL;
   8    2       END PROGRAM;

   9    1       BUFF$IN: PROCEDURE (CHAR) EXTERNAL;
  10    2       DECLARE CHAR    BYTE;
  11    2       END BUFF$IN;

  12    1       BUFF$OUT: PROCEDURE BYTE EXTERNAL;
  13    2       END BUFF$OUT;

                /*****************************************************************
                 *                    LOAD THE INTERRUPT TABLE                   *
                 *****************************************************************/

  14    1       LOAD$INT$TABLE: PROCEDURE PUBLIC;
```

```
PL/M-86 COMPILER    INTMOD
```

```
15   2      CALL SET$INTERRUPT (40H, DEBOUNCE$TIME);
16   2      CALL SET$INTERRUPT (43H, RECEIVE$TIME);
17   2      CALL SET$INTERRUPT (44H, RXD);
18   2      CALL SET$INTERRUPT (45H, TXD);
19   2      CALL SET$INTERRUPT (46H, SCAN$TIME);
20   2      CALL SET$INTERRUPT (47H, LP);

21   2      END LOAD$INT$TABLE;

            /*****************************************************************
             *                    INTERRUPT ROUTINES                         *
             *****************************************************************/


            /*****************************************************************
             *                SET SCAN DELAY FLAG FALSE                      *
             *****************************************************************/

22   1      SCAN$TIME: PROCEDURE INTERRUPT 46H;

23   2      ENABLE;
24   2      SCAN$DELAY=FALSE;
25   2      OUTPUT(CMD$3)=88H;              /*Output end for nested mode*/
26   2      END SCAN$TIME;

            /*****************************************************************
             *              SET DEBOUNCE DELAY FLAG FALSE                    *
             *****************************************************************/

27   1      DEBOUNCE$TIME: PROCEDURE INTERRUPT 40H;
28   2      DEBOUNCE$DELAY=FALSE;
29   2      OUTPUT(CMD$3)=88H;
30   2      END DEBOUNCE$TIME;

            /*****************************************************************
             *               SET RECEIVE DELAY FLAG FALSE                    *
             *****************************************************************/

31   1      RECEIVE$TIME: PROCEDURE INTERRUPT 43H;
32   2      ENABLE;
33   2      RECEIVE$DELAY=FALSE;
34   2      OUTPUT(CMD$3)=88H;
35   2      END RECEIVE$TIME;

            /*****************************************************************
             *                READ SERIAL RECEIVE BUFFER                     *
             *****************************************************************/

36   1      RXD: PROCEDURE INTERRUPT 44H;

37   2      DECLARE
                 STATUS          BYTE,
                 CHAR            BYTE;
38   2      CHAR=INPUT(RX$BUFF);
39   2      OUTPUT(RECEIVE$TIMER)=70;          /* REINITIALIZE RECEIVE TIME OUT */
40   2      STATUS=INPUT(STATUS$REG) AND 0FH;
```

```
PL/M-86 COMPILER     INTMOD

  41   2          IF STATUS<>0
                     THEN
  42   2               CALL ERROR (STATUS),

  43   2          ELSE IF CHAR=ESC
                          THEN
  44   2                    CALL PROGRAM;
                     ELSE
  45   2               CALL BUFF$IN (CHAR);
  46   2       OUTPUT(CMD$3)=88H;
  47   2       END RXD;

            /*************************************************************
             *                SEND A BYTE TO THE LINE PRINTER            *
             *************************************************************/

  48   1       LP: PROCEDURE INTERRUPT 47H;
  49   2       ENABLE;
  50   2       OUTPUT(PORT$2)=BUFF$OUT;
  51   2       OUTPUT(CMD$3)=88H;
  52   2       END LP;

            /*************************************************************
             *                SEND A BYTE TO THE SERIAL PORTS            *
             *************************************************************/

  53   1       TXD: PROCEDURE INTERRUPT 45H;
  54   2       DECLARE
                      MESSAGE BASED MESSAGE$PTR (1) BYTE,
                      I       BYTE;
  55   2       ENABLE;
  56   2       I=MESSAGE(J),
  57   2           IF I<>0
                      THEN OUTPUT(TX$BUFF)=I;
  59   2           ELSE OUTPUT(RST$INT)=TRANSMIT$INT;
  60   2           J=J+1;
  61   2           OUTPUT(CMD$3)=88H;
  62   2       END TXD;

  63   1       END INT$MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 01BDH    445D
     CONSTANT AREA SIZE = 0078H    120D
     VARIABLE AREA SIZE = 0003H      3D
     MAXIMUM STACK SIZE = 0022H     34D
     181 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
PL/M-86 COMPILER     BUFFMOD



SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE BUFFMOD
OBJECT MODULE PLACED IN :F1:BUFF.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:BUFF.SRC


                /*****************************************************************
                 *                                                               *
                 *    BUFFER MODULE:  INSERTS AND REMOVES CHARACTERS FROM FIFO    *
                 *                    REPORTS SERIAL RECEIVE ERRORS AND          *
                 *                    RE-PROGRAMS SERIAL PORTS                   *
                 *                                                               *
                 *****************************************************************/

                $DEBUG
  1             BUFF$MOD: DO;
                $NOLIST

  3    1        DECLARE
                        MESSAGE$PTR     POINTER         PUBLIC,
                        J               BYTE            PUBLIC,
                        OK(1)           BYTE            EXTERNAL,
                        BREAK(1)        BYTE            EXTERNAL,
                        PARITY(1)       BYTE            EXTERNAL,
                        FRAME(1)        BYTE            EXTERNAL,
                        OVER$RUN(1)     BYTE            EXTERNAL,
                        SERIAL$FORMAT(1)BYTE            EXTERNAL,
                        PORT$PTR        BYTE            EXTERNAL,

                        FIFO(8192)      BYTE,
                        IN$POINTER      WORD            PUBLIC,
                        OUT$POINTER     WORD            PUBLIC,
                        BUFFER$STATUS   BYTE            PUBLIC;

                /*****************************************************************
                 *                      INSERT CHARACTER INTO FIFO               *
                 *****************************************************************/

  4    1        BUFF$IN: PROCEDURE (CHAR) PUBLIC;
  5    2        DECLARE
                        CHAR    BYTE;

  6    2        IF ((BUFFER$STATUS=EMPTY) AND ((CHAR=LF) OR (CHAR=CR)))
                    THEN
  7    2              DO;
  8    3                OUTPUT(SET$INT)=HANDSHAKE$INT; /* Enable two-wire handshake interrupt */
  9    3                BUFFER$STATUS=INUSE;
 10    3                OUTPUT(PORT$2)=0;              /* Output NULL character to get
                                                            the interrupt started */
 11    3              END;

 12    2        FIFO(IN$POINTER)=CHAR;    /* Put CHAR into FIFO and increment pointer */
 13    2        IN$POINTER=((INPOINTER+1) AND 1FFFH);

 14    2        IF (((IN$POINTER+4) AND 1FFFH)=OUT$POINTER) /* If the buffer is full, stop reception */
                    THEN
 15    2              DO;  /* Send CTS FALSE, and light up buffer-full LED */
```

```
PL/M-86 COMPILER    BUFFMOD

 16   3              OUTPUT(PORT$1)=((INPUT(PORT$1) OR 40H) AND 7FH);
 17   3              BUFFER$STATUS=FULL;
 18   3             END;
 19   2      END BUFF$IN;

             /*********************************************************************
              *                      REMOVE CHARACTER FROM FIFO                   *
              *********************************************************************/

 20   1      BUFF$OUT: PROCEDURE BYTE PUBLIC;
 21   2      DECLARE  CHAR   BYTE;
 22   2      CHAR=FIFO(OUT$POINTER);
 23   2      OUT$POINTER=((OUT$POINTER+1) AND 1FFFH);
 24   2      IF OUT$POINTER=IN$POINTER  /* If the buffer is EMPTY disable the output to LP */
                THEN
 25   2           DO;
 26   3             DISABLE;
 27   3             BUFFER$STATUS=EMPTY;
 28   3             OUTPUT(RST$INT)=HANDSHAKE$INT;
 29   3             ENABLE;
 30   3           END;

             /*  If the buffer is ready to fill up again then send CTS TRUE */

 31   2      ELSE  IF ((BUFFER$STATUS=FULL) AND (((OUT$POINTER-200) AND 1FFFH)=IN$POINTER))
                    THEN
 32   2                DO; /* Turn off buffer-full LED and turn on CTS */
 33   3                   OUTPUT(PORT$1)=((INPUT(PORT$1) AND 0BFH) OR 80H);
 34   3                   BUFFER$STATUS=INUSE;
 35   3                 END;
             RETURN CHAR;

 37   2      END BUFF$OUT;

             /*********************************************************************
              *                  SEND ERROR MESSAGE TO SERIAL PORT                *
              *********************************************************************/

 38   1      ERROR: PROCEDURE (STATUS) PUBLIC;
 39   2              DECLARE STATUS   BYTE,
                         MESSAGE  BASED  MESSAGE$PTR(1) BYTE;

 40   2              IF (STATUS AND 02H)>0
                        THEN
 41   2                   STATUS=2;
 42   2      ELSE    IF (STATUS AND 04H)>0
                        THEN
 43   2                   STATUS=3;
 44   2      ELSE    IF (STATUS AND 08H)>0
                        THEN
 45   2                   STATUS=4;
 46   2      ELSE    IF (STATUS AND 01H)>0
                        THEN
 47   2                   STATUS=1;
                 DO CASE STATUS;
 49   3                  ;
 50   3                  MESSAGE$PTR=@FRAME(0);
```

```
PL/M-86 COMPILER     BUFFMOD



  51   3                  MESSAGE$PTR=@OVER$RUN(0);
  52   3                  MESSAGE$PTR=@PARITY(0);
  53   3                  MESSAGE$PTR=@BREAK(0);
  54   3              END;

  55   2              J=1;          /* Point to second character in string */
  56   2              OUTPUT(SET$INT)=TRANSMIT$INT;
  57   2              OUTPUT(TX$BUFF)=MESSAGE(0);
  58   2      END ERROR;

              /*****************************************************************
               *                 RELOAD SERIAL PORT CONFIGURE BYTE             *
               *****************************************************************/

  59   1      PROGRAM: PROCEDURE PUBLIC;
  60   2              DECLARE TEMP    BYTE,
                              CHAR    BYTE;

  61   2              DO WHILE (INPUT(STATUS$REG) AND 40H)=0; /* Wait for next byte */
  62   3              END;

  63   2              CHAR=INPUT(RX$BUFF);

  64   2              IF CHAR=0       /* If second byte is 0, exit program mode */
                         THEN
  65   2                     DO;
  66   3                         OUTPUT(RECEIVE$TIMER)=70;
  67   3                         CALL BUFF$IN (CHAR);
  68   3                         RETURN;
  69   3                     END;

  70   2              TEMP=(CHAR AND 0FH);

  71   2              DO WHILE (INPUT(STATUS$REG) AND 40H)=0;
  72   3              END;

  73   2              TEMP=(INPUT(RXBUFF) AND 0FH) OR SHL(TEMP,4);

  74   2              SERIAL$FORMAT (SHR(PORT$PTR,3))=TEMP;
  75   2      END PROGRAM;


  76   1      END BUFF$MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 01E4H    484D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 200BH   8203D
     MAXIMUM STACK SIZE = 000AH     10D
     189 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

```
PL/M-86 COMPILER     PON_MOD

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE PON_MOD
OBJECT MODULE PLACED IN :F1:PON.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:PON.SRC


                $DEBUG

                /*************************************************************************
                 *                                                                       *
                 *       POWER ON INITIALIZATION OF THE LINE PRINTER MULTIPLEXER          *
                 *                                                                       *
                 *************************************************************************/

   1            PON_MOD: DO;

                $NOLIST


   3    1       DECLARE BUFFER$STATUS   BYTE    EXTERNAL,
                        IN$POINTER      WORD    EXTERNAL,
                        OUT$POINTER     WORD    EXTERNAL,
                        PORT$PTR        BYTE    EXTERNAL,
                        SERIAL$FORMAT(8)BYTE    EXTERNAL;


   4    1       POWER$ON: PROCEDURE PUBLIC;

   5    2       DECLARE I               BYTE;


   6    2       DISABLE;

                                /*  INITIALIZE THE MUART */

   7    2       OUTPUT(CMD$1)=01000011B;        /*8086 MODE, FREQ=1KHz, 1 STOP BIT, &
                                                    7 BITS/CHARACTER*/
   8    2       OUTPUT(CMD$2)=10110100B;        /*ODD PARITY, SYSTEM CLOCK=1.024 MHz, &
                                                    9600 bps*/
   9    2       OUTPUT(CMD$3)=01111111B;        /*CLEAR CMD$3 REGISTER*/
  10    2       OUTPUT(CMD$3)=10110001B;        /*RESET, INTERRUPT ACKNOWLEDGE ENABLE, &
                                                    NESTED INTERRUPT MODE*/
  11    2       OUTPUT(MODE)=10000101B;         /*CASCADE TIMERS 35 FOR THE
                                                    RECEIVE$TIME$OUT TIMER, BYTE OUTPUT MODE*/

  12    2       OUTPUT(PORT$1$CTRL)=11111000B;  /*PORT 1: RTS=INPUT, THE REST ARE OUTPUTS*/

  13    2       OUTPUT(PORT$1)=11000000B;       /*POINT TO THE FIRST PORT, CTS IS FALSE,
                                                    AND BUFFER IS NOT FULL*/

                      /* INITIALIZE FLAGS, VARIABLES, AND ARRAYS */

  14    2       BUFFER$STATUS=EMPTY;
  15    2       IN$POINTER=0;   OUT$POINTER=0;
  17    2       PORT$PTR=0;

  18    2               DO I=0 TO 7;
```

```
PL/M-86 COMPILER    PON_MOD

  19   3              SERIAL$FORMAT(I)=10010100B;     /* ON POWER-UP ALL EIGHT SERIAL PORTS
                                                       DEFAULT TO 9600 bps, ODD PARITY, AND
                                                       7 BITS/CHARACTER*/
  20   3              END;
  21   2        END POWER$ON;
  22   1        END PON_MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 0058H     88D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0001H      1D
     MAXIMUM STACK SIZE = 0002H      2D
     98 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

APPENDIX B
LISTING OF THE WRITE PROGRAM

```
PL/M-80 COMPILER

ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE WRITEMOD
OBJECT MODULE PLACED IN :F1:WRITE.OBJ
COMPILER INVOKED BY:  :F2:PLM80 :F1:WRITE.SRC


                $DEBUG
  1             WRITE$MOD: DO;

                /*****************************************************************
                 *                                                               *
                 *    WRITE PROGRAM:  READS A FILE FROM A DISK AND COPIES        *
                 *                    IT TO SERIAL CHANNEL 2 ON THE MDS.         *
                 *                                                               *
                 *    SYNTAX OF WRITE:  WRITE :DEVICE:NAME.EXTENSION             *
                 *                                                               *
                 *****************************************************************/

  2   1         DECLARE  LIT             LITERALLY       'LITERALLY',
                       USART$DATA       LIT             '0F6H',
                       USART$STATUS     LIT             '0F7H',
                       RTS              LIT             '20H',
                       TXEN             LIT             '01H',
                       RXE              LIT             '04H',
                       CR               LIT             '0DH',
                       LF               LIT             '0AH',
                       TAB              LIT             '09H',
                       SP               LIT             '20H',
                       ESC              LIT             '1BH',
                       FORM$FEED        LIT             '0CH';

  3   1         DECLARE  AFT$IN          ADDRESS,
                       FILENAME(15)     BYTE,
                       STATUS           ADDRESS,
                       BUFFER(32000)    BYTE,
                       ACTUAL           ADDRESS,
                       CHAR$COUNT       ADDRESS,
                       BYE(42)          BYTE    INITIAL
                ('WROTE ',0,0,0,0,0,0,0,0,0,0,0,0,0,0,' TO THE LINE PRINTER',0AH,0DH),
                       I                ADDRESS,
                       J                BYTE;

                /*****************************************************************
                 *                  EXTERNAL SYSTEM.LIB PROCEDURES               *
                 *****************************************************************/

  4   1         OPEN:
                       PROCEDURE (AFTNPTR,FILE,ACCESS,MODE,STATUS) EXTERNAL;
  5   2                 DECLARE (AFTNPTR,FILE,ACCESS,MODE,STATUS) ADDRESS;
  6   2         END OPEN;

  7   1         READ:
                       PROCEDURE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) EXTERNAL;
  8   2                 DECLARE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
  9   2         END READ;

 10   1         WRITE:
```

```
PL/M-80 COMPILER

                        PROCEDURE (AFTN, BUFFER, COUNT, STATUS) EXTERNAL;
 11    2                DECLARE (AFTN, BUFFER, COUNT, STATUS) ADDRESS;
 12    2        END WRITE;

 13    1        CLOSE:
                        PROCEDURE (AFTN, STATUS) EXTERNAL;
 14    2                DECLARE (AFTN, STATUS) ADDRESS;
 15    2        END CLOSE;

 16    1        ERROR:
                        PROCEDURE (ERRNUM) EXTERNAL;
 17    2                DECLARE (ERRNUM) ADDRESS;
 18    2        END ERROR;

 19    1        EXIT:
                        PROCEDURE EXTERNAL;
 20    2        END EXIT;

                /*****************************************************************
                 *              WAIT UNTIL USART TRANSMITTER IS READY            *
                 *****************************************************************/

 21    1        TXRDY:
                        PROCEDURE;
 22    2                DO WHILE ( (INPUT(USART$STATUS) AND 01H) = 0 );
 23    3                END;
 24    2        END TXRDY;

                /*****************************************************************
                 *                        BEGIN MAIN PROGRAM                     *
                 *****************************************************************/
 25    1        BEGIN:
                        STATUS=0;

 26    1                CALL READ(1,.FILENAME,15,.ACTUAL,.STATUS);  /* Read in file and path name */

 27    1                IF STATUS <> 0
                             THEN
 28    1                        GO TO DONE;

 29    1                CALL OPEN(.AFT$IN,.FILENAME,1,0,.STATUS); /* Open up the file */

 30    1                IF STATUS <> 0
                             THEN
 31    1                        GO TO DONE;

 32    1        REPEAT:
                        CALL READ(AFT$IN,.BUFFER,32000,.ACTUAL,.STATUS);

 33    1                IF STATUS <> 0
                             THEN
 34    1                        GO TO DONE;

 35    1                CHAR$COUNT=0;  /* CHAR$COUNT keeps track of the tab columns in each line */

 36    1                OUTPUT(USART$STATUS)= RTS OR TXEN;
```

PL/M-80 COMPILER

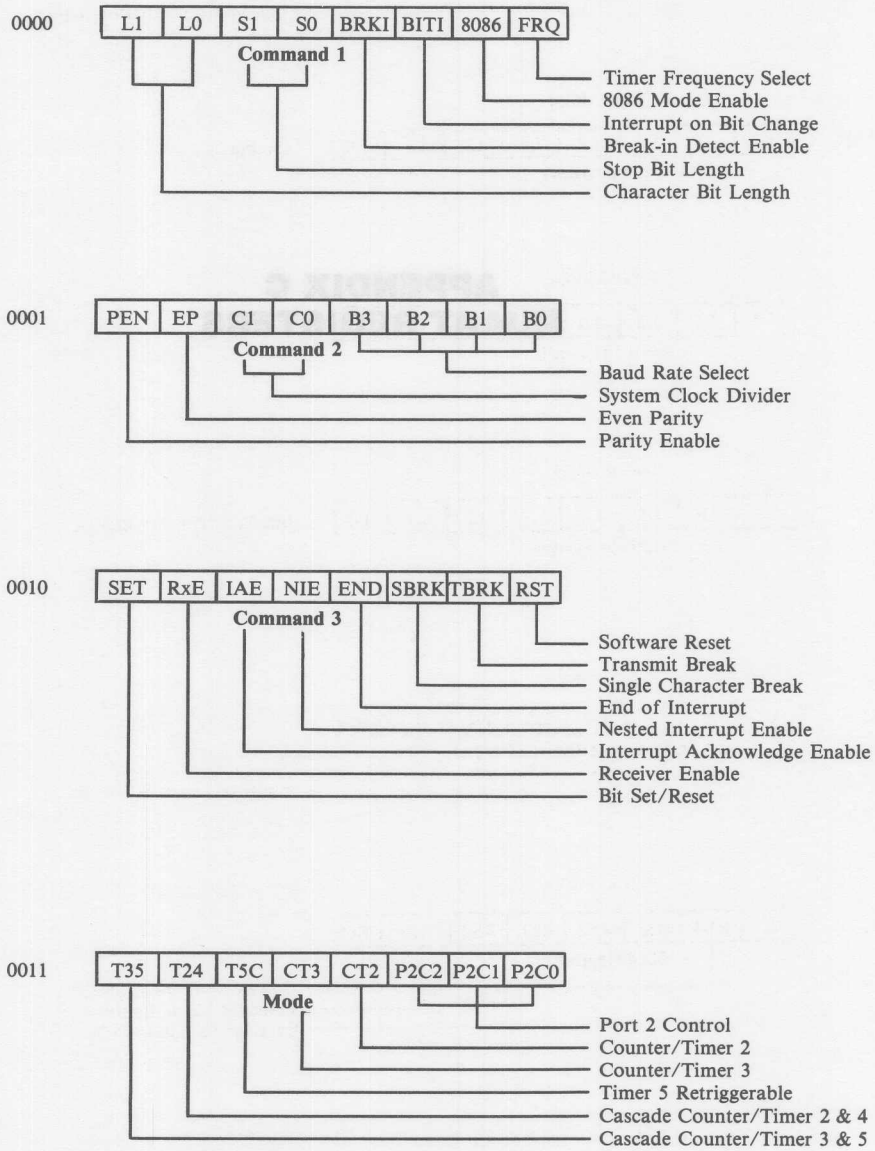
```
37   1              IF BUFFER(0)=FORM$FEED   /* If the first character is a form feed
                                                remove it. Form feeds are inserted at the
                                                end of a file */
                         THEN
                            DO;
38   1                        BUFFER(0)=00H;
39   2                        CHAR$COUNT=-1;
40   2                     END;
41   2
42   1              DO I = 0 TO (ACTUAL - 1);
43   2                     IF  (BUFFER(I)=TAB)  /* Replace TAB characters with the
                                                   appropriate number of spaces */
                               THEN
                                  DO;
44   2
45   3                                CALL TXRDY;
46   3                                OUTPUT(USART$DATA)=SP;
47   3                                CHAR$COUNT=CHAR$COUNT+1;
48   3                                DO WHILE ((CHAR$COUNT AND 0007H)<>0);
49   4                                      CALL TXRDY;
50   4                                      OUTPUT(USART$DATA)=SP;
51   4                                      CHAR$COUNT=CHAR$COUNT+1;
52   4                                END;
53   3                             END;
                            ELSE
54   2                             IF BUFFER(I)=ESC /* If outputting ESC, then output a
                                                      0 next so the LPM does not get
                                                         re-programmed */
                                        THEN
55   2                                   DO J=0 TO 1;
56   3                                      CALL TXRDY;
57   3                                      OUTPUT(USART$DATA)=0;
58   3                                   END;
                                  ELSE /* If the character is not an ESC or TAB then output it */
59   2                                DO;
60   3                                   CALL TXRDY;
61   3                                   OUTPUT(USART$DATA)=BUFFER(I);
62   3                                   IF (BUFFER(I)>1FH AND BUFFER(I)<>7FH)
                                              THEN                    /* Only increment CHAR$COUNT
                                                                         for printable characters */
63   3                                        CHAR$COUNT=CHAR$COUNT+1;
64   3                                   IF ( (BUFFER(I)=CR) OR (BUFFER(I)=LF) )
                                              THEN                    /* Reset CHAR$COUNT for CR or LF */
65   3                                      CHAR$COUNT=0;
66   3                                END;
67   2              END;
68   1              IF ACTUAL = 32000 /*If the file is more than 32K, get some more data */
                         THEN
69   1                     GO TO REPEAT;
70   1              CALL TXRDY;                    /* Terminate file with CR, LF, and FF */
71   1              OUTPUT(USART$DATA)=CR;
72   1              CALL TXRDY;
```

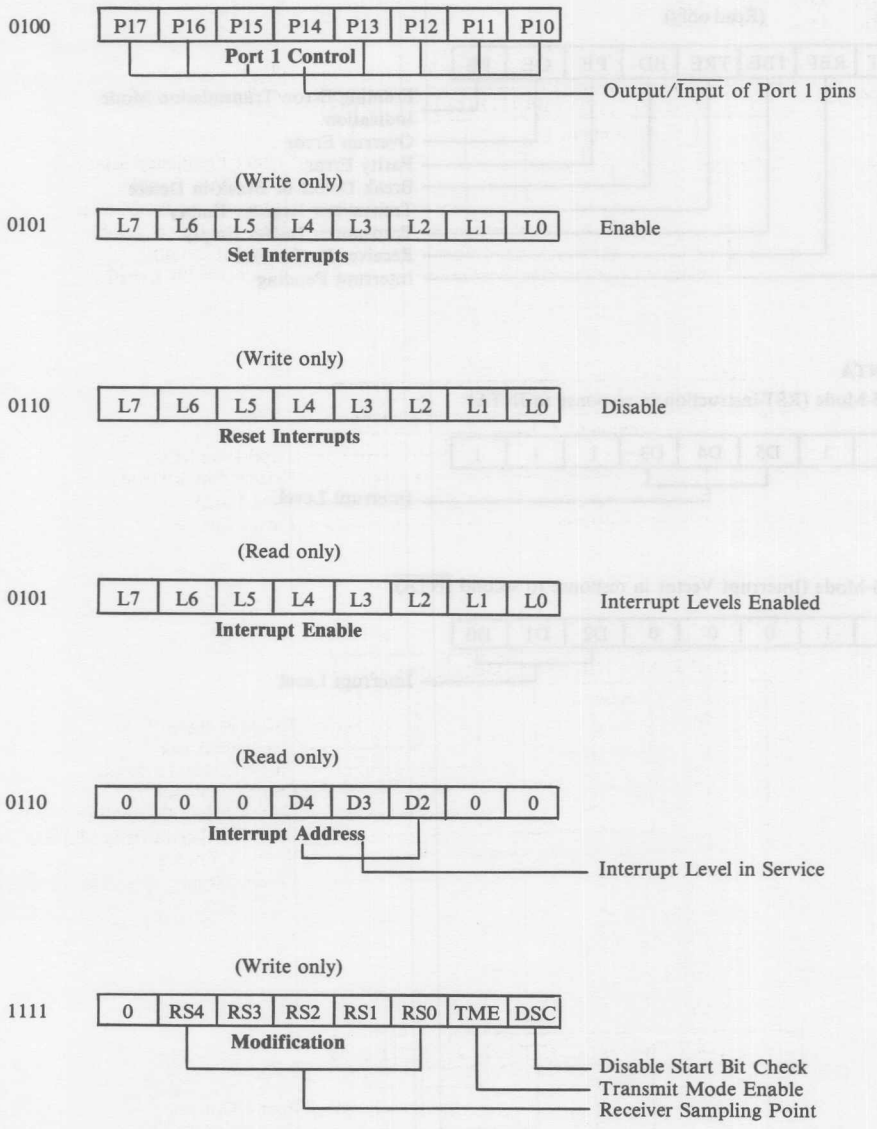
```
PL/M-80 COMPILER

  73   1              OUTPUT(USART$DATA)=LF;
  74   1              CALL TXRDY;
  75   1              OUTPUT(USART$DATA)=FORM$FEED;

  76   1              OUTPUT(USART$STATUS)=RXE OR TXEN; /* Shut off RTS */

  77   1              CALL CLOSE (AFT$IN,.STATUS);

  78   1              DO I=0 TO 14;              /* Output sign off message */
  79   2                   IF FILENAME(I)=CR
                            THEN
  80   2                        GO TO SKIP;
  81   2                   BYE(I+5)=FILENAME(I);
  82   2              END;

  83   1       SKIP:  CALL WRITE(0,.BYE,42,.STATUS);

  84   1              GO TO NEXT;

  85   1       DONE:
                      CALL ERROR(STATUS);

  86   1       NEXT:
                      CALL EXIT;

  87   1       END WRITE$MOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 0209H    521D
     VARIABLE AREA SIZE = 7D44H  32068D
     MAXIMUM STACK SIZE = 0008H      8D
     191 LINES READ
     0 PROGRAM ERRORS

END OF PL/M-80 COMPILATION
```

APPENDIX C
MUART REGISTERS

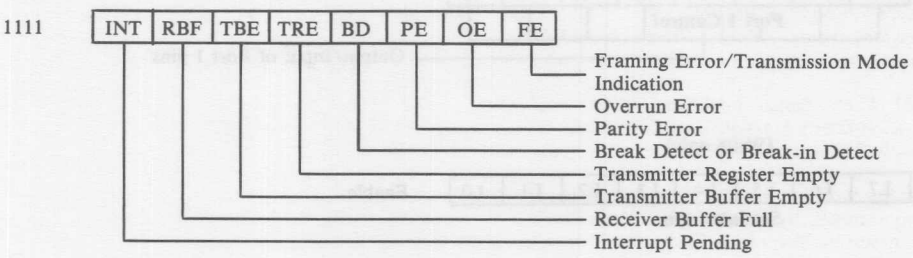
8085 Mode: AD3 AD2 AD1 AD0

8086 Mode: AD4 AD3 AD2 AD1



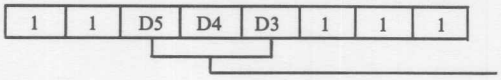

0100
P17 P16 P15 P14 P13 P12 P11 P10
Port 1 Control
Output/Input of Port 1 pins
(Write only)
0101
L7 L6 L5 L4 L3 L2 L1 L0
Enable
Set Interrupts
(Write only)
0110
L7 L6 L5 L4 L3 L2 L1 L0
Disable
Reset Interrupts
(Read only)
0101
L7 L6 L5 L4 L3 L2 L1 L0
Interrupt Levels Enabled
Interrupt Enable
(Read only)
0110
0 0 0 D4 D3 D2 0 0
Interrupt Address
Interrupt Level in Service
(Write only)
1111
0 RS4 RS3 RS2 RS1 RS0 TME DSC
Modification
Disable Start Bit Check
Transmit Mode Enable
Receiver Sampling Point

Status Register (Read only)

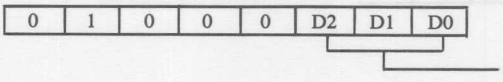


Response to INTA

8085-Mode (RST-instruction in response to $\overline{\text{INTA}}$)



8086-Mode (Interrupt Vector in response to second $\overline{\text{INTA}}$)



REFERENCES

1. *8256 User's Manual,* SIEMENS.
2. *Application Note, AP-59: Using the 8259A Programmable Interrupt Controller,* Intel Corp., CA.
3. *MCS-85 User's Manual,* Intel Corp., CA.
4. *iAPX 86,88 User's Manual,* Intel Corp., CA.
5. *Printronix 300 Applications Manual,* Printronix, CA.
6. *High Integration LSI for Low Cost Asynchronous Communications,* By Charles Yager Professional Program Session Record 13, Electro/82.

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, CA 95051 (408) 987-8080

INTEL INTERNATIONAL, Brussels, Belgium; Tel. (02) 661 07 11

INTEL JAPAN k.k., Ibaraki-ken; Tel. 029747-8511.

PRINTED IN USA T-1429/10K/0583/HAR/JOS CG